ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

24

Basic

AIA

SINCLAIR AMSTRAD COMMODORE IBM



EDICIONES SIGLO CULTURAL

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

24

Basic

EDICIONES SIGLO CULTURAL

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática
JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño y maquetación:
BRAVO-LOFISH.

Dibujos:
JOSE OCHOA Y ANTONIO PERERA.

Tomo XXIV. **Basic.** AULA INFORMATICA APLICADA (AIA).
JESUS BOCHO. Licenciado en Informática.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta (Iberia Mart-I). Teléf.: 810 52 13. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América. 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-083-9.
ISBN de la obra: 84-7688-018-9.

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 9.603-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta (Iberia Mart-I). Teléf.: 810 52 13. 28020 Madrid.

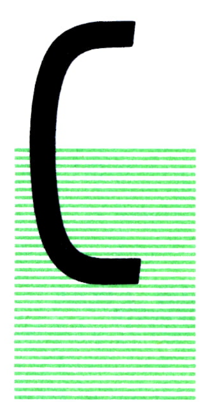
Abril, 1987
P.V.P. Canarias: 365,-

INDICE

Los programas que aparecen en este libro funcionan en los ordenadores:

IBM–PC, XT, AT y compatibles.
AMSTRAD–464, 664, 6128, 1512.
SINCLAIR–SPECTRUM 48 K, 128 K, PLUS, PLUS 2.
MSX–Todos los modelos.
COMMODORE–CBM 64 y CBM 128.

LOS PROGRAMAS 1



UANDO nosotros queremos que nuestro ordenador haga algo tenemos que decírselo previamente de una manera que él lo pueda entender. Para ello, lo que debemos hacer es, en primer lugar, determinar qué pasos debe realizar para conseguir lo que pretendemos y a continuación escribir esos pasos en lenguaje que el ordenador sea capaz de entender. Estos lenguajes que nuestro ordenador es capaz de comprender son los llamados LENGUAJES DE PROGRAMACION. Hay numerosos lenguajes de programación, siendo los más conocidos el BASIC, el PASCAL, el COBOL ó el FORTRAN. En este libro nos vamos a dedicar a estudiar el BASIC, aunque se va a tratar de dar una visión general de la programación que pueda ser muy útil a la hora de estudiar cualquier otro lenguaje.

Un lenguaje de programación estará compuesto, pues, por una serie de instrucciones, que serán las distintas órdenes que se le pueden dar al ordenador. De esta manera, nosotros deberemos obtener los pasos que tiene que dar el ordenador para hacer lo que deseamos como un conjunto de las distintas órdenes que el ordenador puede realizar. Estas órdenes son las suficientes como para poder conseguir que el ordenador haga todo lo que nosotros queramos. Basta con que sepamos «desmenuzar» lo que queremos hacer en ese conjunto de instrucciones que el ordenador es capaz de entender. Así, una vez que hayamos obtenido el conjunto de pasos que debe dar el ordenador para que realice aquello que deseamos, debemos escribir estos pasos de una manera ordenada en el lenguaje de programación con el que estemos trabajando. Este conjunto de instrucciones ordenadas en un lenguaje de programación, escritas con el fin de que el ordenador las ejecute una detrás de otra, es lo que llamamos programa.

Las instrucciones se ordenarán de distintas maneras en los diferentes lenguajes. En BASIC cada instrucción de un programa debe llevar un número delante. Cuando el ordenador ejecute el programa (realice una a una

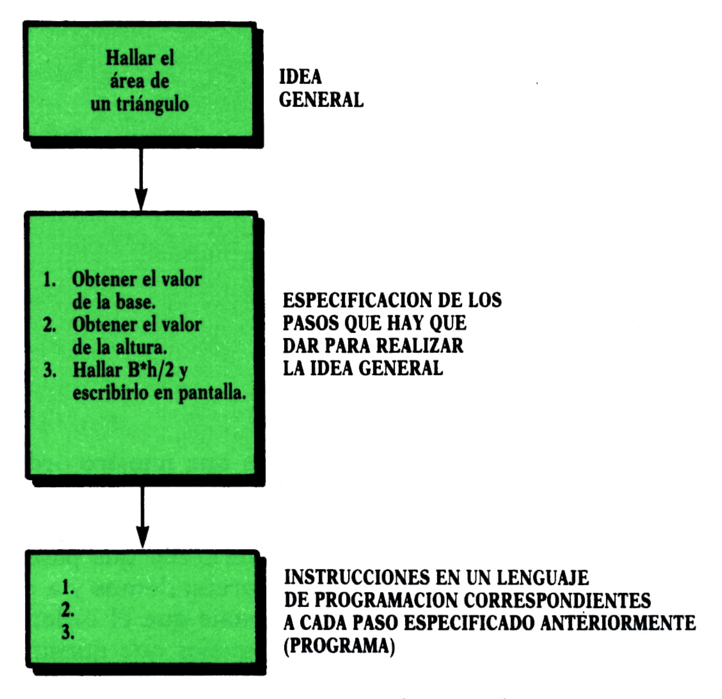


Fig. 1. Esquema general de la realización de un programa.

las órdenes correspondientes a las instrucciones que le hemos dado) empezará por ejecutar aquella que lleve un número mas bajo y continuará ejecutando sucesivamente por orden de menor a mayor número. De esta manera, si en un programa que ya tenemos, queremos introducir una instrucción entre medias de las existentes (bien porque se nos haya olvidado o bien porque queramos añadir alguna cosa nueva), deberemos introducirla con un número intermedio entre las que queremos que sean anterior y posterior en el orden de ejecución. Por esta razón normalmente las instrucciones en BASIC se numeran de 10 en 10.

Una vez que tenemos determinadas las instrucciones que compondrán nuestro programa debemos introducirlas todas juntas en el ordenador cuando se encuentre en modo «comando». (El modo «comando» es aquél en el que está el ordenador después de haber entrado en BASIC.) Después de haberlas introducido todas escribiremos el comando RUN, por el que el ordenador nos las ejecutará todas juntas, una detrás de otra, siguiendo el orden de los números en que las hemos puesto. Esto es lo que se llama ejecutar un programa.

Cuando un programa tiene una única instrucción podemos escribirla directamente sin número y el ordenador nos realizará su ejecución inmediatamente después de escribirla. Esto es lo que se llama ejecución en

modo inmediato. El objetivo de este libro será precisamente estudiar las distintas órdenes que podemos necesitar dar al ordenador y cómo escribir estas órdenes en lenguaje BASIC.

A continuación se presenta como ejemplo un programa BASIC. Aunque el lector aún no entienda el significado de las instrucciones que aparecen, sí puede darse una idea de la estructura según lo explicado anteriormente.

```
10 REM PROGRAMA QUE CALCULA SI UN NUMERO ES PRIMO
20 REM ******************************************
30 CLS
40 INPUT "Introduce un número";X
50 IF (X/2)=INT(X/2) AND X>2 THEN PRINT"no es numero primo": GOTO 100
60 FOR CON=3 TO INT(SQR(X)) STEP 2
70 IF (X/CON)=INT(X/CON) THEN PRINT"no es numero primo":GOTO 100
80 NEXT CON
90 PRINT "es primo"
100 INPUT "Quieres seguir jugando(s/n)";A$
110 IF A$="S" OR A$="s" THEN GOTO 40
```

NOTA: Recuerda que después de escribir cada línea en el ordenador se debe pulsar RETURN, INTRO o ENTER.

Si en una línea de programa (con su número correspondiente) colocamos la palabra REM, a continuación podemos escribir lo que queramos, porque el ordenador, al ejecutar el programa, no va a hacer caso de estas líneas. Esto nos permite introducir comentarios sobre lo que hace el programa, como se puede ver en el ejemplo.

En una misma línea de programa (con el mismo número por tanto) podemos introducir varias instrucciones BASIC si las separamos por dos puntos. Cuando demos orden de ejecución al programa, las ejecutará una detrás de otra en el orden en que estén.

EJERCICIOS:

1. ¿Qué diferencia hay entre el modo inmediato y el modo programa en BASIC?

2. ¿Para qué sirven los números de instrucción dentro de un programa BASIC?

3. Especifica los pasos que tendría que dar un programa para resolver una ecuación de segundo grado.

RESPUESTAS:

1. El modo inmediato es útil para operaciones y cálculos que vayamos a hacer de forma ocasional y que se puedan especificar como una sola ins-

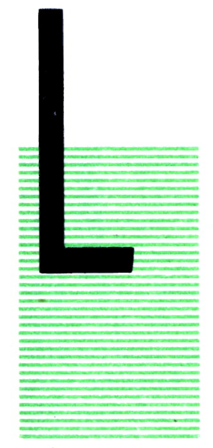
trucción en BASIC. Escribiremos un programa siempre que lo que vayamos a hacer requiera más de una instrucción o sea algo que vayamos a utilizar en distintas ocasiones, con lo que nos convendrá meter un programa para que se quede grabado en la memoria del ordenador y podamos ejecutarlo siempre que queramos escribiendo únicamente RUN.

2. Los números de instrucción ordenan las líneas de un programa. La ejecución comienza por la sentencia de número más bajo y continúa en secuencia.

3. a) *Obtener los valores de los coeficientes de la ecuación.*
b) *Resolver la fórmula de cálculo de las raíces X1 y X2.*
c) *Escribir por pantalla o impresora el resultado de los cálculos anteriores.*

NOTA: Las respuestas a los ejercicios propuestos en cada tema vendrán en algunos de ellos, mientras que en otros se dejarán como ejercicios propuestos al lector.

EL LENGUAJE BASIC 2



A palabra BASIC viene de las iniciales en inglés de la frase:

«Beginers All-purpose Symbolic Instruction Code».

Es un lenguaje de programación que, como su nombre indica, está especialmente indicado para principiantes.

Fue diseñado por John G. Kemeny y Thomas E. Kurtz, en 1963, para introducir a sus alumnos del colegio Dartmouth en los lenguajes de programación.

Es un lenguaje universal, ya que tenemos la posibilidad de utilizarlo prácticamente en todos los microcomputadores.

La potencia del lenguaje fue creciendo en sucesivas versiones a lo largo de los años. Su evolución, sin embargo, no ha sido estándar, y así nos encontramos con que, a excepción de las instrucciones básicas del lenguaje (las que veremos en este primer número), cada ordenador tiene para algunas operaciones, como sobre todo gráficos, instrucciones diferentes. Incluso en ese grupo de instrucciones básicas que pretenden ser estándar, hay algunas variantes en los distintos equipos.

El BASIC tiene un gran número de adeptos, ya que resulta fácil de aprender y puede resultar un buen primer paso para aprender a programar. También se trata de un lenguaje de uso general que sirve tanto para trabajos administrativos como para aplicaciones científicas.

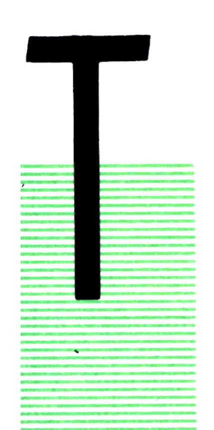
Sin embargo, tiene una serie de inconvenientes también que hacen que no sea empleado en el desarrollo de aplicaciones complejas, para las que resultan más útiles otros lenguajes, como el PASCAL o el COBOL. Estos inconvenientes son principalmente la rigidez a que nos someten los números de línea que no permiten programar de manera estructurada, como ocurre con otros lenguajes en los que se puede programar a base de ir ensamblando distintos módulos de programa que se han realizado independientemente.

Otro ligero inconveniente resulta el hecho de que el BASIC utilice intérprete. El intérprete significa, de modo sencillo, que cuando nosotros hemos terminado de introducir todas las instrucciones de nuestro programa, damos la orden de ejecución (RUN) y el ordenador ejecuta directamente las instrucciones que hemos puesto en el orden correspondiente. El resto de los lenguajes de programación utilizan, sin embargo, lo que se llama compilador. El compilador consiste en que cuando nosotros hemos terminado de escribir las instrucciones de nuestro programa tenemos, antes de mandarlo ejecutar, que dar la orden al computador de que transforme las instrucciones que hemos puesto en el lenguaje de programacion correspondiente, en instrucciones de código máquina. Después de realizar esto, le daremos la orden de ejecutar estas instrucciones de código máquina (que harán evidentemente lo mismo que nuestro programa). De esta manera, cuando queremos ejecutar un MISMO programa varias veces únicamente tenemos que mandar ejecutar las instrucciones de código máquina, cuya ejecución es bastante más rápida que las instrucciones del lenguaje de programación correspondiente que ejecutará en el caso del BASIC todas las veces que queramos que ejecute el mismo programa.

Como resumen, podemos decir que el BASIC es un lenguaje de programación fácil de aprender y que puede ser una buena manera de aprender a programar. Además, nos puede servir para realizar cualquier tipo de problema que queramos resolver con el ordenador, siempre que éstos no sean grandes aplicaciones para las que lógicamente nos resultaría más sencillo usar otros lenguajes que nos permitan mayor estructuración, aunque siempre se podrían hacer en BASIC, ya que es un lenguaje que tiene las mismas posibilidades que cualquiera de los otros lenguajes de programación.

LOS DATOS Y LAS VARIABLES 3

DEFINICION



TODOS los programas que nosotros hagamos necesitarán manejar datos. Los datos son ese conjunto de valores que el programa va a utilizar para obtener los resultados deseados, la información que usará.

Cada lenguaje de programación se caracteriza, entre otras cosas, por los distintos tipos de datos que es capaz de manejar. Atendiendo a este criterio diremos que hay lenguajes más ricos que otros. El BASIC se puede decir que es un lenguaje pobre en cuanto a tipos de datos. A continuación vamos a estudiar los dos grandes tipos de datos que nos ofrece, que, de todas formas, nos dan grandes posibilidades de tratamiento.



TIPOS DE DATOS EN BASIC

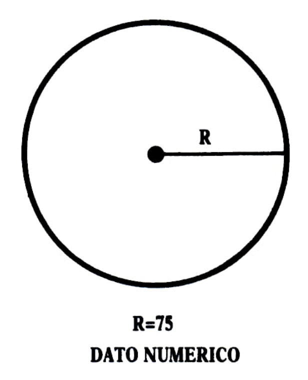
Cualquier lenguaje debe ofrecernos capacidad para representar, por lo menos, dos tipos fundamentales y diferenciados de información:

— Numérica.
— Alfanumérica.

Los datos numéricos son, como su nombre indica, aquellos números que son empleados para cálculos matemáticos.

Los datos alfanuméricos son toda la información que maneja nuestro programa y que está compuesta por las letras del alfabeto, los números del 0 al 9, así como el resto de símbolos especiales que posee nuestro ordenador, como interrogaciones, puntos, comas, etc.

Conviene resaltar desde un principio la diferencia fundamental que existe entre un número, expresado como dato alfanumérico, y el mismo número expresado como dato numérico.



DOMICILIO: CALLE DE GOYA, N.º 75

DATO ALFANUMERICO

Vamos a analizar los dos ejemplos siguientes:

domicilio: calle de Goya, número 75

radio del círculo: 75

El número 75 que estaría empleando un programa que manejase esta información tiene una utilización muy diferente: en el primer caso, se trata de una información descriptiva que no será utilizada en cálculos. A lo sumo nos implicará un orden dentro de la calle. Podremos comprobar entre dos direcciones de la misma calle quién vive más al principio. El número 75 que expresa el radio del círculo es, sin embargo, muy diferente. El programa lo utilizará muy posiblemente en cálculos matemáticos como sumas, restas, multiplicaciones y demás operaciones aritméticas. En el primer caso se trata de datos alfanuméricos, mientras que en el segundo se trata de datos numéricos.

Los lenguajes de programación (el BASIC entre ellos) nos van a proporcinar métodos para transformar datos alfanuméricos en numéricos, y viceversa.



LAS CONSTANTES Y LAS VARIABLES

Hasta ahora hemos visto que un programa necesita manejar una serie de información que puede ser de dos grandes tipos, pero no sabemos cómo el programa representa y guarda estos datos. Este problema nos lo va a solucionar el concepto de constante y variable.

a) Las constantes:

Las constantes son la representación que el programa hace de los datos que utiliza. Así, los datos numéricos los representará el programa me-

diante el número correspondiente. El 1 o el 1543 son ejemplos de constantes numéricas (forma de representar los datos numéricos correspondientes). Cuando el dato numérico es un número decimal se representa mediante un punto. Por ejemplo: 5.6, 234.67.

Las constantes alfanuméricas estarán formadas por cualquier combinación de letras, números o cualquier otro tipo de caracteres. La única particularidad que tienen es que en todo programa BASIC deben ir encerradas entre comillas. De aquí se deduce que el único carácter que no puede formar parte de un dato alfanumérico son las comillas. Así, «calle», «Goya» o «¿n1?» son ejemplos de representaciones válidas en un programa Basic de datos alfanuméricos.

b) Las variables:

Los datos (las constantes que acabamos de ver) tienen que ser guardados por el programa en algún lugar (el programa los tiene que tener «apuntados» en algún sitio para utilizarlos cuando los necesite). Donde el ordenador guarda todos sus datos es en la memoria. Cuando nosotros hacemos un programa (escribimos las órdenes que queremos darle al ordenador para que las ejecute) debemos darle órdenes de introducir datos en memoria o traer datos que tiene guardados en memoria para realizar las operaciones pertinentes con ellos.

Las variables son precisamente las CASILLAS de la memoria donde el ordenador guarda los datos. En cada variable, que se corresponderá con una «casilla» de memoria, el ordenador guardará un dato. Hay variables numéricas (en las que se guarda un dato numérico) y variables alfanuméricas (en las que se guarda un dato alfanumérico.

Para poder distinguir una «casilla» donde tenemos guardado un valor de otra donde tenemos guardado otro valor diferente, debemos darlas a cada una de estas «casillas» (variables) nombres distintos. Estos nombres, que daremos en nuestro programa tantos como número de variables distintas necesitemos utilizar para guardar los datos que necesitará el programa, deben estar formados por un grupo de letras o números que, dependiendo del tipo de ordenador en que se trabaje, podrán tener una mayor o menor longitud máxima.

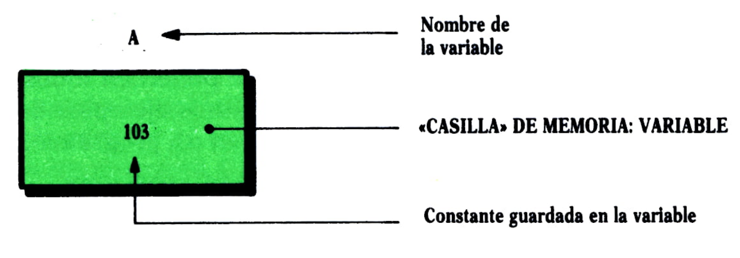


Fig. 2.

Las variables alfanuméricas, para que el ordenador sepa que en esa «casilla» lo que tiene que guardar es un dato alfanumérico, deben llevar al final de su nombre el símbolo $.

— Ejemplos:

A, B, NUM son ejemplos de variables numéricas.
A$, C$, DIA$, CALLE$ son ejemplos de variables alfanuméricas.



LA ARITMETICA EN BASIC

Cuando hablábamos de los datos numéricos decíamos que eran datos que podían ser utilizados en un programa para realizar distintos cálculos matemáticos con ellos. Vamos a ver qué operaciones podemos mandar hacer al ordenador con los datos en un programa y cómo se representan estas operaciones.

Con los datos numéricos podemos realizar:

Suma: Se escribe mediante el signo +.
Resta: Mediante el signo –.
Multiplicación: Mediante el *.
División normal: Mediante el /.
División entera: Mediante el \.
Resto de la división: Mediante la palabra MOD.
Potenciación: Mediante el signo ^

La operación «división entera» consiste en obtener el cociente de la división de dos números truncando su parte decimal.

Cuando en una instrucción de un programa queremos realizar una operación aritmética con dos o más datos, debemos escribir los datos (expresados como una constante o como la variable donde está guardado el dato), poniendo entre medias el signo de la operación correspondiente. También se pueden utilizar los paréntesis siguiendo las reglas matemáticas de su colocación, pero teniendo una cosa en cuenta: en Basic no se ponen corchetes en ningún nivel de paréntesis, sino que se ponen en todos los niveles los paréntesis. A continuación se ponen distintos ejemplos de las operaciones aritméticas que podríamos encontrar en un programa BASIC:

R2+7, F*R+9, ((786/3)–(432-RAD) MOD 5)*X

(las letras son nombres de variables donde están guardados los datos con los que queremos operar).

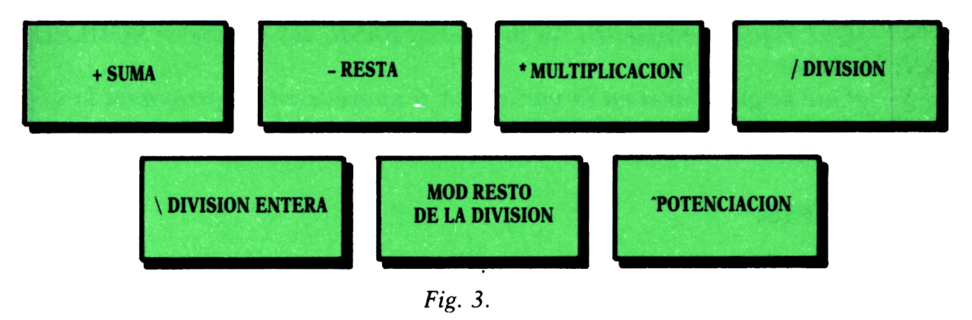
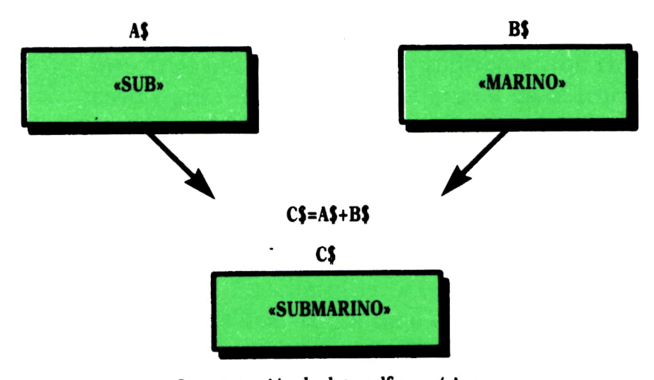


Fig. 3.

Con los datos alfanuméricos:

En lo que se refiere a operaciones aritméticas propiamente dichas, con los datos alfanuméricos únicamente podemos realizar la concatenación, consistente en unir dos cadenas de caracteres distintas (datos alfanuméricos), colocando en el resultado la segunda a continuación de la primera. Se representa mediante el símbolo + y en la figura se muestra un ejemplo de cómo se realizaría una concatenación que indicásemos nosotros en un programa.



Concatenación de datos alfanuméricos

Con los datos alfanuméricos se pueden realizar otro tipo de operaciones utilizando funciones que se verán en el capítulo correspondiente.

EJERCICIOS:

1. ¿Qué es una variable? ¿Qué tipos conoces?

2. Si en la variable A$ tenemos guardado «12» y en la variable B$ «36» y hacemos en un programa C$=A$+B$ ¿qué nos guardará el ordenador en C$?

3. Di de qué tipo serían las variables cuyos nombres fueran los siguientes: A, NUM, H$, G, X$.

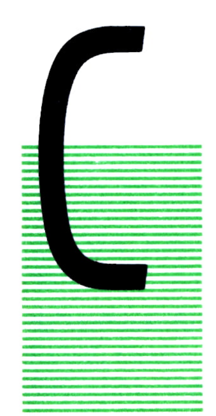
4. *¿Qué representarían en un programa BASIC las palabras: NUMERO y «NUMERO»?*

5. *¿Qué se guardaría en la variable A si aparece en un programa la expresión: A=((4*9)–(9–7))3 MOD 3?*

RESPUESTAS:

2. *«1236».*
4. a) *NUMERO: Nombre de una variable.*
   b) *«NUMERO»: Dato alfanumérico.*
5. *2.*

LOS COMANDOS DEL BASIC 4



OMO ya se dijo en el primer capítulo, cuando nosotros encendemos el ordenador y entramos en el intérprte BASIC se dice que estamos en el modo «comando». Este modo significa que el ordenador está preparado para recibir o bien líneas de programa que irá almacenando en su memoria con el número que nosotros les pongamos o bien los llamados comandos. Estos comandos son palabras que nosotros escribimos en una línea y que realizan una función concreta que lleva asignada cada uno. Por medio de ellos nos evitamos la necesidad de conocer el sistema operativo de la máquina con la que trabajamos, ya que las funciones que realizan son equivalentes a las operaciones del sistema operativo que tendríamos que realizar.

Debido a que el BASIC no está normalizado, los comandos pueden variar de una máquina a otra. En cualquier caso, siempre encontraremos comandos similares. La sintaxis correcta de cada comando debe consultarse en el manual del BASIC que estemos utilizando, ya que junto al nombre del comando que aquí especificaremos puede haber algún tipo de argumento o datos auxiliares.

A continuación se pone una lista de los comandos más importantes ordenados por orden alfabético:

AUTO. Genera los números de línea del programa automáticamente. Cuando estamos introduciendo un programa nos escribe, al terminar de introducir una instrucción y pulsar RETURN, en la línea siguiente el número de la siguiente instrucción. El incremento de una línea a otra se elige al dar el comando, generalmente se hace de 10 en 10.
CLEAR. Pone a cero todas las variables del programa que tenemos en ese momento introducido en memoria.
CONT. Continúa la ejecución de un programa detenido por un STOP.

DELETE. Permite borrar líneas de un programa. Se pueden borrar con este comando una o varias.

Ejemplo:

```
10 INPUT B
20 INPUT A
30 LET AR=B*A/2
40 PRINT "EL AREA ES"
50 PRINT AR
```

tras pulsar DELETE 20-40 nos quedaría:

```
10 INPUT B
50 PRINT AR
```

EDIT. Trae una línea del programa para poder hacer modificaciones sobre ella.

FILES. Hace un listado en pantalla de los nombres de los ficheros contenidos en el disco.

KILL. Borra del disco o disquete el fichero especificado.

LIST. Lista en pantalla las líneas que queramos del programa que tenemos en memoria. Como normalmente el programa puede no cavernos entero en pantalla, deberemos seleccionar un rango de líneas para que sean visualizadas.

LLIST. Lista todo el programa o parte por la impresora.

LOAD. Carga en memoria un programa que tengamos almacenado en el disco. Cuando queremos conservar un programa que hemos metido en el ordenador y, por tanto, se encuentra almacenado en su memoria debemos grabarlo en un disco o disquete porque los programas almacenados en memoria al dejar de trabajar con el BASIC (apagar el ordenador o salir al sistema operativo) se borran. En el disco el programa se grabará como un fichero más. Un poco más adelante se describirá el comando para grabar un programa en el disco.

MERGE. Permite fusionar un programa que está en el disco con el que actualmente tenemos en memoria.

NEW. Borra el programa que tenemos en memoria y sus variables.

RENUM. Renumera las líneas de un programa. Esto es especialmente útil cuando al añadir líneas intermedias nos hemos encontrado sin huecos para intercalar más.

Ejemplo:

El programa:

```
10 INPUT B
12 INPUT A
14 LET AR=B*A/2
16 PRINT "EL EREA ES"
18 PRINT AR
```

tras dar el comando **RENUM** nos quedaría:

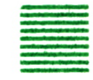
```
10 INPUT B
20 INPUT A
30 LET AR=B*A/2
40 PRINT "EL EREA ES"
50 PRINT AR
```

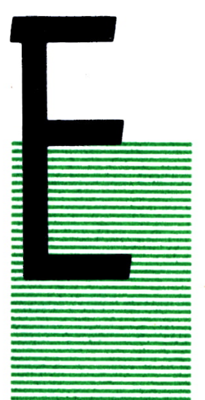
RUN. Ejecuta el programa que tenemos en memoria.
SAVE. Graba un programa que tenemos en memoria en un disco.
SYSTEM. Abandona el BASIC devolviendo el control al sistema operativo.

Para equipos con unidad de cinta se utilizan los mismos comandos u otros muy parecidos que los descritos para los discos.

LA ESCRITURA DE LOS DATOS 5

INTRODUCCION





EN los primeros capítulos hemos visto una serie de temas necesarios para entender los conceptos de programa, lenguaje de programación y saber desenvolverse mínimamente a la hora de comenzar a trabajar en BASIC en un ordenador.

A partir de aquí vamos a comenzar a ver las instrucciones y estructuras que se nos pueden plantear a la hora de darle al ordenador las órdenes de lo que queremos hacer y veremos cómo se escriben estas instrucciones en el lenguaje BASIC formándonos nuestro programa.



Fig. 1.

La primera instrucción que se nos plantea es escribir datos bien por pantalla o por impresora. Indudablemente, cuando nosotros en un programa realizamos las operaciones que sean, siempre tendremos que decirle al ordenador que nos escriba aquello que queramos obtener como resultado, porque sería estúpido que después de hacer una serie de operaciones no nos sacara ningún resultado, ya que entonces el programa no nos serviría de nada.

Así, todos los programas deberán tener POR LO MENOS UNA INSTRUCCION de escribir datos por pantalla o impresora.



¿QUE PODEMOS ESCRIBIR?

Las cosas que le podemos mandar escribir al ordenador mediante una instrucción de un programa se pueden resumir en tres grandes tipos:

1. Frases literales:

Serán aquellas frases que nosotros queramos que nos escriba tal y como las ponemos nosotros. Por ejemplo, cuando queremos que antes de escribirnos el resultado del área de una figura que nos ha calculado nos ponga: EL AREA ES.

2. Datos guardados en variables.
3. Resultado de una expresión aritmética.



LA INSTRUCCION EN BASIC: PRINT

La instrucción que tenemos en BASIC para escribir datos es PRINT.

Cuando escribimos un programa y queremos utilizarlo debemos ponerla de la siguiente manera:

— N. de instr. PRINT aquello que queremos que nos escriba.

A continuación de la palabra PRINT debemos poner lo que queremos que el programa nos escriba de la siguiente forma:

a) Frases literales: Se debe poner la frase entre comillas. Por ejemplo: 10 PRINT «EL AREA ES».

Cuando se ejecutara esta instrucción nos escribiría en la pantalla EL AREA ES.

b) Datos guardados en una variable: Se pone a continuación de PRINT el nombre de la variable. Por ejemplo: 10 PRINT NUM.

Cuando se ejecute esta instrucción escribirá el número que haya guadado en NUM.

c) Expresiones aritméticas: Se pone a continuación de PRINT la expresión aritmética tal y como se explicó en el capítulo 3. La expresión puede estar compuesta tanto por variables como constantes con sus respectivos operadores. Por ejemplo:

10 PRINT (A*2)/D

Cuando se ejecute esta instrucción se escribirá por pantalla el resultado de la operación indicada.

Cada instrucción PRINT que utilicemos escribe lo que en ella pongamos en una línea distinta. Cuando queramos escribir varias cosas en una misma línea debemos ponerlas en una instrucción PRINT separadas por ;. Por ejemplo:

10 PRINT «EL AREA ES»;B*H/2

Si queremos que después de una instrucción PRINT no nos cambie de línea y en la siguiente continúe en la misma, bastará con que pongamos un ; al final del PRINT.

Si separamos las distintas cosas que ponemos en un PRINT por una coma en vez del punto y coma cuando nos lo escriba nos lo hará con una separación de 6 blancos. Poniendo un punto y coma nos deja un blanco.

Para escribir por impresora en vez de por pantalla, debemos cambiar PRINT por LPRINT.

A continuación se ofrecen dos programas en los que se utiliza la instrucción PRINT:

```
10 REM PROGRAMA QUE CALCULA EL AREA DE UN TRIANGULO DE BASE 4 Y ALTURA 10
20 REM ********************************************************************
30 PRINT "EL AREA ES"
40 PRINT 4*10/2
```

```
10 REM OBTENCION DEL RESULTADO DE UNA EXPRESION
20 REM ****************************************
30 PRINT "EL RESULTADO ES";((345*6)/3 +23)*(45+8)
```

PODEMOS MANDAR ESCRIBIR DONDE QUERAMOS

Si ejecutamos los programas puestos anteriormente o cualquier otro observaremos que lo que le mandamos escribir en la instrucción PRINT lo escribe en la línea siguiente a la última que haya escrita en la pantalla y a partir de la primera columna siempre.

Puede darse el caso, sin embargo, de que nosotros queramos que nos escriba una determinada cosa que le digamos mediante la instrucción PRINT en una línea y columna determinada de la pantalla. Por ejemplo, queremos que nos escriba un mensaje en el centro de la pantalla. Para poder realizar esto, tenemos una instrucción en BASIC: LOCATE.

La instrucción LOCATE se escribe de la siguiente manera:

LOCATE f, c

siendo *f* y *c* los números de fila y columna respectivamente donde queremos que el cursor de la pantalla se sitúe para comenzar a escribir en la próxima instrucción PRINT que venga.

Así, en un programa podríamos poner:

20 LOCATE 10, 20

que significaría que la próxima instrucción PRINT se escribirá en la fila 10 columna 20; o también podríamos poner:

20 LOCATE A, B

y querrá decir que la próxima PRINT se escribirá en la fila cuyo número está guardado en la variable A y la columna cuyo número está guardado en la variable B. En el próximo capítulo veremos cómo guardar valores en las variables.

Cuando se utiliza la instrucción LOCATE hay que tener en cuenta que si mandamos escribir en una fila o columna de la pantalla donde ya hay algo escrito (que pueden ser, por ejemplo, líneas del programa que hayamos mandado visualizar anteriormente), se escribirá encima, pudiendo quedar confuso al mezclarse unas cosas con otras. Para que esto no ocurra se aconseja en estos casos anteponer la instrucción de borrar toda la pantalla, que en la mayoría de los ordenadores suele ser CLS, aunque para algunas máquinas cambia.

Para el uso de la instrucción LOCATE se aconseja consultar previamente los manuales de la máquina que se vaya a usar para ver el número de líneas y columnas de que consta la pantalla.

SPECTRUM	COMMODORE	AMSTRAD	MSX	IBM-MS-DOS
PRINT AT fila, col.	No existe	LOCATE col., fila	LOCATE fila, col.	LOCATE fila, col.

Variantes de la instrucción LOCATE en los distintos equipos

A continuación se adjuntan dos programas en los que se puede ver más claramente el empleo de LOCATE.

```
10 REM OBTENCION DEL RESULTADO DE UNA EXPRESION ESCRIBIENDOLO EN LA FILA 10
20 REM ******************************************************************
25 CLS
27 LOCATE 10,1
30 PRINT "EL RESULTADO ES";((345*6)/3 +23)*(45+8)
```

```
10 REM TITULOS EN PANTALLA
20 REM ******************
30 CLS
40 LOCATE 5,30:PRINT "CLASE DE INFORMATICA"
50 LOCATE 10,26:PRINT "HORARIO   : DE 10 A 14 HORAS"
```

En una instrucción LOCATE también podemos poner como valores de fila y columna expresiones aritméticas. En estos casos, la fila o columna donde se situará el cursor será aquella que dé como resultado de la expresión indicada.

EJERCICIOS:

1. Escribir un programa que obtenga en pantalla los siguientes resultados:

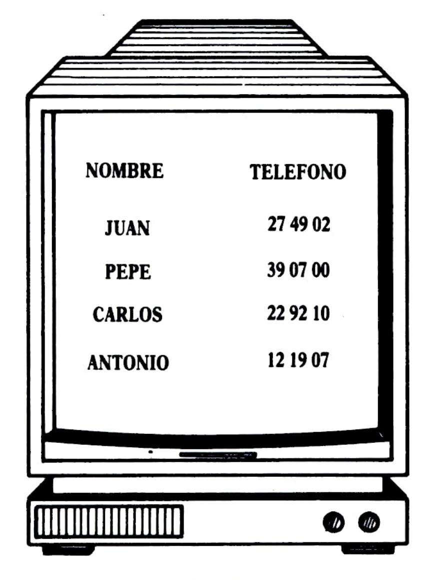


Fig. 2.

*2. Escribir una instrucción que escriba en pantalla: 37*5=, escribiendo a continuación el resultado de la operación.*

3. Escribir un programa que calcule el área de un círculo de radio 5.

4. Escribir un programa que ponga en el centro de la pantalla el nombre de tu ordenador.

RESPUESTAS:

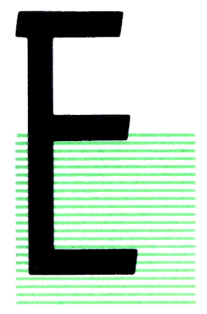
2.

```
10 PRINT "37*5=";37*5
```

ASIGNACION DE VALORES A LAS VARIABLES 6



INTRODUCCION



EN el capítulo 3 vimos lo que era una variable. Uno de los primeros problemas que se nos plantean a la hora de utilizar las variables en un programa es el de guardar en ellas los valores (constantes). Para realizar esto, existe evidentemente una instrucción que es la que vamos a estudiar en este capítulo.



LA INSTRUCCION LET

Cuando en un programa queremos guardar un valor en una variable debemos utilizar en BASIC la instrucción LET. Esta instrucción se escribe de la siguiente manera:

LET nombre de la variable = valor que queremos guardar

Indudablemente, delante de la palabra LET debe ir el número de instrucción que le corresponda, como en todas las instrucciones BASIC.

Visto esto, serían ejemplos de instrucciones LET:

```
10 LET N=345
```

Esta instrucción nos guardaría en la variable N el número 345.

También podríamos poner:

```
30 LET B=(456*76)/4 ó
40 LET NUM=NUM+1
```

Estas nos guardarían en las variables B y NUM los resultados de las expresiones aritméticas colocadas a la derecha del igual y escritas de acuerdo con las reglas BASIC para expresiones aritméticas.

La palabra LET no es obligatorio ponerla en todos los ordenadores, excepto en el SPECTRUM, donde hay que ponerla siempre. De ahí, que en mu-

chos programas que aparezcan en este libro habrá instrucciones LET en las que no aparezca la palabra. Aquellos usuarios que programen en el SPECTRUM deberán incluirla en todos los casos. De esta forma, las instrucciones anteriores nos quedarían:

10 N=345 y
40 NUM=NUM+1

A continuación se ofrecen dos programas en los que se usa la instrucción LET. Obsérvese que el del cálculo del área es el mismo que hacíamos en el capítulo anterior, con la diferencia de que ahora guardamos previamente los datos en una variable.

```
10 REM AREA DE UN TRIANGULO DE BASE 4 Y ALTURA 10
20 REM ****************************************
30 LET B=4
40 LET A=10
50 PRINT "EL AREA ES";B*A/2
```

```
10 REM CALCULO DE FORMULAS
20 REM ******************
30 LET A=4
40 LET A=A+1
50 PRINT A
60 LET A=2*A+3
70 PRINT A
80 LET B=A-2
90 PRINT A,B
100 LET B=(A-B)^2
110 PRINT A,B
```



LAS CLASES DE VARIABLES NUMERICAS

Cuando en un programa los números que vamos a guardar en una misma variable van a ser siempre enteros, podemos indicárselo al programa poniendo al final del nombre de la variable (sin dejar espacio en blanco) el símbolo tanto por ciento (%).

De igual manera, si van a ser decimales de precisión simple, podemos poner el símbolo !, y si van a ser decimales de precisión doble, el símbolo P_t.

El hecho de realizar esto aumenta la velocidad de ejecución.

Cuando hagamos esto, en las instrucciones LET deberemos poner al final de todas las constantes que asignemos a las variables el mismo símbolo que le hemos puesto a la variable.

Ejemplo:

40 A%=A%+1%

EJERCICIOS:

1. Escribe un programa que calcule el área de un círculo de radio 5 guardando previamente el valor del radio en una variable.

2. Escribe un programa que escriba en el centro de la pantalla el nombre de tu ordenador, guardándolo previamente en una variable.

3. Escribe un progrma que te escriba un número y su doble utilizando instrucciones LET.

RESPUESTAS:

3.

```
10 LET A=18
20 PRINT A
30 LET A=A*2
40 PRINT A
```

CUANDO EL PROGRAMA NECESITA PEDIRNOS DATOS 7



INTRODUCCION

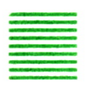
EN los programas vistos en el capítulo anterior para la obtención del área de diversas figuras geométricas podemos observar que el programa nos calcula siempre que le ejecutamos el área de una figura cuyos valores de base y altura en el caso del triángulo, por ejemplo, son siempre los mismos. Cuando queremos hallar el área de un triángulo de valores distintos tenemos que cambiar una o varias instrucciones del programa.

Probablemente, al ver los programas anteriores, el lector se haya preguntado si no habría alguna posibilidad de que el programa calculase el área de cualquier triángulo sin necesidad de tenerlo que cam-



Fig. 1.

biar. Para ello sería necesario que cada vez que ejecutásemos el programa, éste nos pidiera los valores de la base y la altura y calculase el área del triángulo a partir de estos valores. Esto es posible gracias a la instrucción que vamos a estudiar en este capítulo.



LA INSTRUCCION INPUT

La instrucción que realiza lo explicado anteriormente es la instrucción INPUT. Su formato general es:

INPUT «mensaje»; nombre de variable

Cuando el programa llega a ejecutar una instrucción INPUT lo que hace es:

a) Escribir en la pantalla el mensaje que hemos colocado entre comillas en la instrucción (lo escribe sin comillas).

b) Quedarse esperando a que nosotros tecleemos un número en el caso de que el nombre de la variable que hemos puesto en la instrucción INPUT corresponda a una variable numérica o esperar a que tecleemos una palabra si el nombre corresponde a una variable alfanumérica.

c) Una vez que hemos tecleado el dato y pulsado RETURN, el programa GUARDARA el dato que hemos metido en la variable que hay en la instrucción INPUT y continuará ejecutando el programa en la siguiente instrucción.

Así, el programa del cálculo del área del triángulo nos quedaría de la siguiente manera:

```
10 REM AREA DEL TRIANGULO
20 REM *****************
30 CLS
40 INPUT "DAME EL VALOR DE LA BASE";B
50 INPUT "DAME EL VALOR DE LA ALTURA";A
60 PRINT "EL AREA ES";B*A/2
```

Cuando ejecutáramos el programa haría lo que se especifica en la figura:

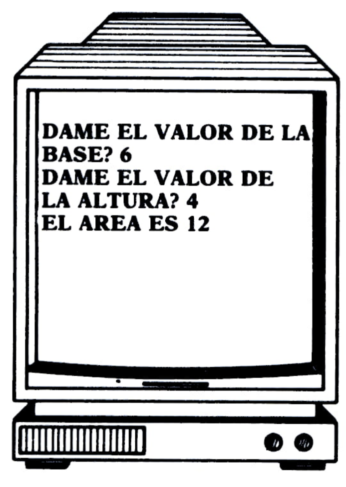


Fig. 2.

Notas auxiliares:

Al final del mensaje que hemos puesto en el INPUT, el programa nos sacará un signo de interrogación cuando durante la ejecución nos pida el dato correspondiente. Si queremos que no nos escriba este signo de interrogación, debemos poner una coma en vez de un punto y coma para separar el mensaje de la variable.

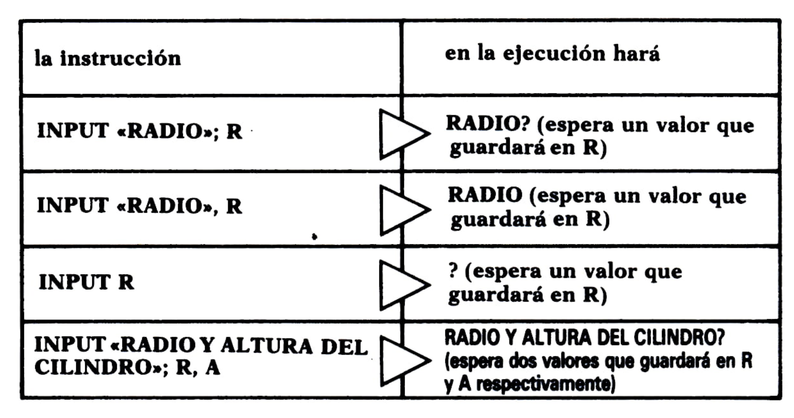
El mensaje no es obligatorio ponerlo. Si no lo ponemos nos sacará cuando se ejecute la instrucción el signo de interrogación y se quedará esperando a que le metamos el dato. De todas formas, es aconsejable ponerlo para saber durante la ejecución del programa qué dato nos está pidiendo.

Si queremos que el programa nos pida varios datos para guardarlos en variables distintas podemos poner una única instrucción INPUT y colocar en ella los nombres de todas las variables separados por comas. Cuando esta instrucción se ejecute se quedará esperando hasta que le metamos tantos datos como variables hayamos puesto. Estos datos debemos introducirlos *todos en una misma línea y separados por comas, no debiendo pulsar RETURN hasta que no hayamos tecleado el último dato.* El programa anterior nos quedaría de la siguiente manera utilizando esto:

```
10 REM AREA DEL TRIANGULO
20 REM *****************
```

```
30 CLS
40 INPUT "DAME EL VALOR DE LA BASE Y LA ALTURA";B,A
60 PRINT "EL AREA ES";B*A/2
```

Cuando en la ejecución de una instrucción INPUT introducimos datos que no son correctos para lo que nos está pidiendo, el programa nos los volverá a pedir, volviéndose a quedar esperando a que se los metamos.



la instrucción	en la ejecución hará
INPUT «RADIO»; R	RADIO? (espera un valor que guardará en R)
INPUT «RADIO», R	RADIO (espera un valor que guardará en R)
INPUT R	? (espera un valor que guardará en R)
INPUT «RADIO Y ALTURA DEL CILINDRO»; R, A	RADIO Y ALTURA DEL CILINDRO? (espera dos valores que guardará en R y A respectivamente)

Distintas posibilidades de la instrucción INPUT

Cuando diéramos la orden de ejecución al programa del último ejemplo, se comportaría de la siguiente forma:

Instrucción 40:

Aparecería en pantalla el mensaje:

TECLEE LA BASE Y LA ALTURA?

Si algún carácter tecleado no es numérico o no tecleamos dos constantes numéricas separadas por una coma, no se aceptará nada de lo tecleado, obteniéndose un mensaje que invita a teclear de nuevo desde el principio. Los valores numéricos tecleados se almacenarán en las variables B y A, respectivamente.

Instrucción 60:

Escribe lo indicado por la instrucción PRINT.

A continuación se ofrecen tres programas ejemplo de empleo de la instrucción INPUT:

```
10 REM CALCULO DEL CUADRADO Y LA RAIZ DE CUALQUIER
   NUMERO
20 REM ****************************************
30 INPUT"DAME UN NUMERO";N%
40 LET C%=N%^2
50 LET R=N%^(1/2)
60 PRINT "EL CUADRADO ES";C%
70 PRINT "LA RAIZ ES";R
```

```
10 REM PROGRAMA QUE PASA DE PESETAS A DOLARES
20 REM ************************************
30 INPUT "A CUANTAS PESETAS ESTA EL DOLAR";CAM
40 INPUT "CANTIDAD QUE QUIERES CAMBIAR ",PES
50 PRINT PES;"PESETAS SON";PES/CAM;"DOLARES"
```

```
10 REM CALCULO DEL VALOR DE UN  POLINOMIO
20 REM ********************************
30 INPUT "INTRODUCE EL VALOR DE X ",X
40 LET P=X^4+3*X^3-X^2+X+1
50 PRINT "EL VALOR DEL POLINOMIO ES";P
```

EJERCICIOS:

1. Escribir un programa que ejecute la fórmula siguiente para cualquier dato de entrada A y B:

*C=A/2 *B2*

2. ¿Qué ocurre si en una sentencia INPUT que pide datos para una variable numérica, tecleamos un valor alfabético?

3. Idea un programa que utilice INPUT, PRINT y LET.

4. Idea un programa que pida los gastos e ingresos de una empresa y calcule el 12% de los beneficios (impuestos).

5. Haz un programa que te pida los datos de la dirección de una persona y los escriba como si se tratara de un sobre.

6. Idea un programa que te pida una palabra latina y te la decline según la primera declinación.

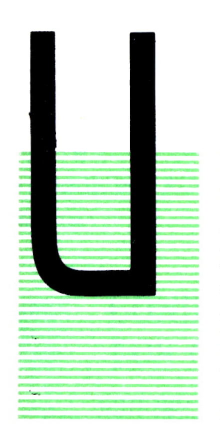
RESPUESTAS:

6.

```
10 REM PRIMERA DECLINACION
20 REM ******************
30 INPUT "PALABRA PARA DECLINAR";PAL$
40 PRINT PAL$
50 PRINT PAL$
60 PRINT PAL$+"M"
70 PRINT PAL$+"E"
80 PRINT PAL$+"E"
90 PRINT PAL$
```

ESTRUCTURAS CONDICIONALES LA TOMA DE DECISIONES 8

INTRODUCCION



NA cuestión que se nos plantea muy a menudo al realizar un programa es cuando queremos que haga una determinada acción (instrucción del programa) únicamente en algunos casos y no siempre. Con lo que hemos visto hasta ahora no podemos conseguir esto, ya que cuando en un programa colocamos una instrucción, siempre que le demos al programa la orden de ejecución, ejecutará todas las instrucciones sean cuales sean las condiciones en cada ocasión (independientemente, por ejemplo, de los valores que haya en determinadas variables).

En este capítulo vamos a ver cómo podemos conseguir que un programa ejecute una o varias instrucciones sólo cuando se cumpla una determinada condición.

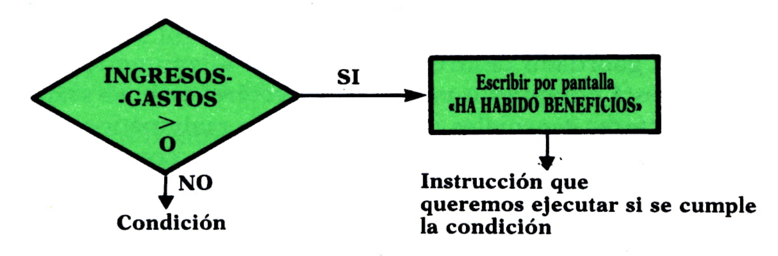


Fig. 1.



LAS CONDICIONES EN UN PROGRAMA

Diremos, en primer lugar, que una condición en un programa es una expresión que puede tener dos posibles soluciones: verdadero o falso; es decir, que se cumpla o no se cumpla.

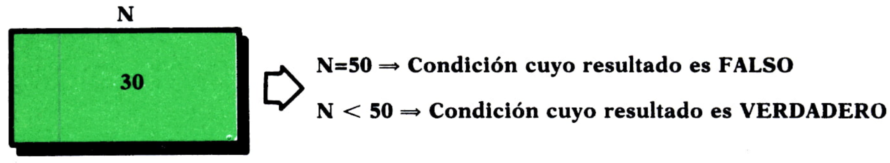


Fig. 2.

Las condiciones en un programa las escribiremos como comparación de:

a) Dos variables.
b) Un valor (constante) con una variable de su mismo tipo.
c) Una expresión aritmética con una variable de su mismo tipo.

Estas comparaciones las podemos hacer mediante los siguientes signos:

=	IGUAL
<>	DISTINTO
<	MENOR
>	MAYOR
<=	MENOR O IGUAL
>=	MAYOR O IGUAL

Así, podrían ser expresiones de comparaciones en un programa:

A=NUM , G>8 , X <= N*5 , N$=«JOSE PEREZ» ,X$= A$+B$

Las variables alfanuméricas también las podemos comparar mediante los símbolos mayor y menor y mayor o igual y menor o igual. Una palabra será menor que otra si ordenada por orden alfabético es anterior y en el caso contrario será mayor.

Si el valor guardado dentro de la variable alfanumérica que estamos comparando está compuesto por otros caracteres que no sean letras (números o caracteres especiales como los signos matemáticos o las interrogaciones), la comparación será igual, de acuerdo con un orden interno que tienen estos caracteres dentro de nuestro ordenador. En la figura se ilustran estos casos:

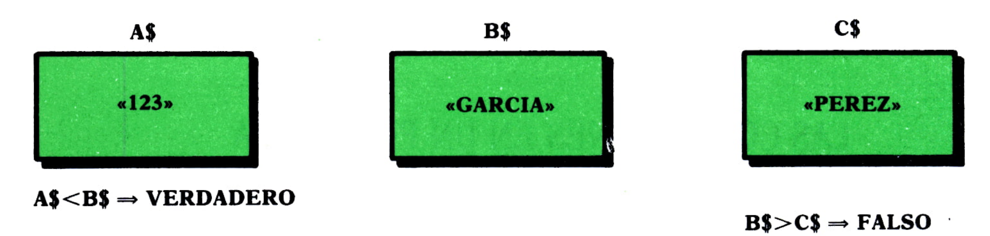


Fig. 3.

El valor «123» es menor que «GARÇIA», porque internamente en nuestro ordenador el número 1 (inicial del primer valor) es anterior a las letras.

Hay que tener en cuenta al trabajar con valores alfanuméricos que las letras mayúsculas son distintas de las minúsculas. Así, por ejemplo, la condición «CASA» = «casa» sería falsa en el supuesto de que comparásemos dos variables alfanuméricas que tuviesen guardados estos valores.

Resumiendo, podemos decir que una condición será verdadera cuando la comparación de las dos variables, o de la variable con un valor o expresión, que hayamos puesto mediante el signo correspondiente, sea cierta en ese momento, mientras que será falsa cuando no sea cierta.

Si en un programa ponemos:

```
10 LET A =20
```

la condición A>30 será falsa, mientras que la condición A<=50 será verdadera.



CONDICIONES MULTIPLES. LOS OPERADORES LOGICOS

En ocasiones queremos utilizar condiciones que estén compuestas por varias comparaciones. Por ejemplo, yo quiero ejecutar una determinada instrucción en un programa cuando se cumpla que A>20 y además B<200. También puedo querer ejecutar una instrucción cuando bien pase que A<=550 o que N$=«garcía». Esto son lo que llamamos las condiciones múltiples y en BASIC y la mayoría de los lenguajes de programación se expresan mediante los operadores lógicos.

Estos operadores lógicos son:

AND: Se pone entre dos condiciones y diremos que la condición resultante es verdadera si se cumplen las dos condiciones entre las que está y falsa en el resto de los casos.

OR: De igual forma se pone entre dos condiciones y diremos que la condición resultante es verdadera si una de ellas es verdadera.

NOT: Se coloca delante de una condición y cambia el resultado de ésta. Si es verdadera, hace que sea falsa, y si es falsa, verdadera.

Los operadores OR y AND funcionarían igual que unos interruptores eléctricos colocados tal y como se indica en la figura.

Los operadores lógicos son fundamentales en situaciones de selección. Si deseamos escribir por pantalla los datos de las personas que cumplan varias condiciones a la vez (por ejemplo, que vivan en una ciudad de-

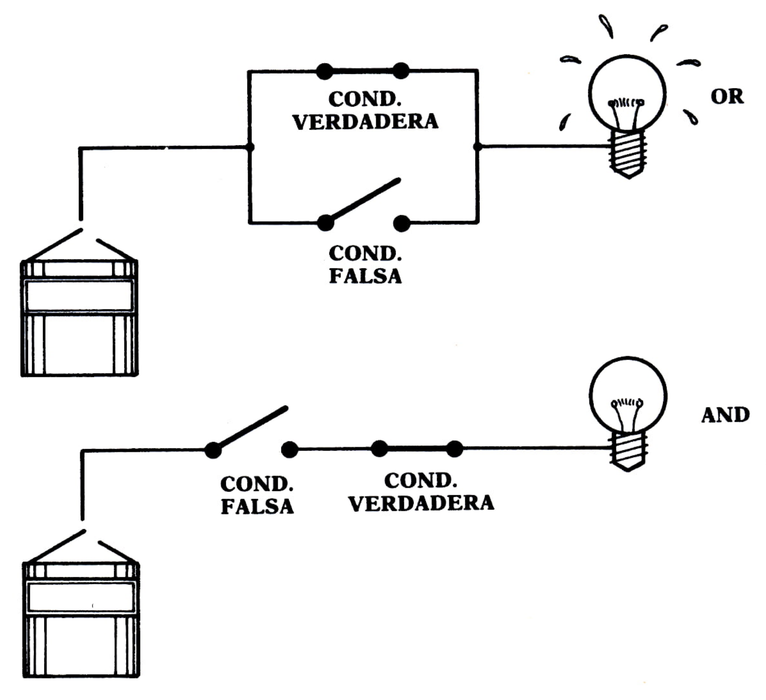


Fig. 4.

terminada y sean mayores de cuarenta años), utilizaremos el operador AND. Para elegir los que cumplen una o las dos condiciones, se emplea el operador OR.

Así, en un programa podríamos ver condiciones de la forma:

A>30 AND B>4444 ,A$="b" OR X>5 ,(A$="G" AND H=6) OR D$> «PEPE»

La última condición será verdadera si ocurre que en A$ hay guardada una «G» y en H un 6, o bien ocurre que la palabra guardada en D$ está alfabéticamente después de «PEPE».



LAS DECISIONES EN BASIC

El BASIC, por medio de la instrucción IF, permite tomar decisiones dentro de un programa.

La instrucción IF se escribe de la siguiente manera:

IF condición THEN cualquier instrucción BASIC

La condición podrá ser cualquiera de las condiciones simples o múltiples vistas en los apartados anteriores y después de la palabra THEN podremos poner cualquier instrucción de BASIC o varias separadas por dos puntos.

Cuando el programa llega a ejecutar una instrucción IF evalúa la condición que hay entre IF y THEN y si es verdadera, el programa ejecutará la instrucción o instrucciones que hay a continuación del THEN. Si la condición fuera falsa, el programa no haría nada y pasaría a la siguiente instrucción.

Ejemplo:

El siguiente programa lee un número desde el teclado. Se obtiene un mensaje diferente según sea par o impar.

```
10 REM PROGRAMA EJEMPLO SENTENCIA IF
20 REM ****************************
30 INPUT "TECLEE UN NUMERO ",N
40 IF(N MOD 2)=0 THEN MENSAJE$="Numero Par"
50 IF(N MOD 2)<>0 THEN MENSAJE$="Numero Impar"
60 PRINT MENSAJE$
```

El operador aritmético MOD nos da el resto de dividir el número almacenado en N entre 2. El resto obtenido se compara con 0. Si el resultado de la comparación es verdadero (caso de número par), se asigna a la variable alfanumérica MENSAJE$ el valor «Numero Par». En caso contrario, se asigna el valor «Numero Impar». En cualquier caso, el programa termina mostrando, por la sentencia PRINT, el mensaje correspondiente en la pantalla.

A continuación se adjunta otro programa de empleo de la instrucción IF.

```
10 REM CALCULO DEL AREA DE 2 FIGURAS GEOMETRICAS
20 REM ****************************************
30 INPUT "BASE";B
40 INPUT "ALTURA";A
50 INPUT "QUE FIGURA ES (TRIANGULO O RECTANGULO)";F$
55 PRINT "EL AREA ES";
60 IF F$="TRIANGULO" THEN PRINT B*A/2
70 IF F$="RECTANGULO" THEN PRINT B*A
```



LA INSTRUCCION GOTO

Vamos a ver en este apartado otra instrucción BASIC muy relacionada con las estructuras condicionales: la instrucción GOTO.

Su formato es el siguiente:

GOTO número de otra instrucción del programa

Cuando un programa llega a ejecutar una instrucción GOTO, lo que hace es saltar a ejecutar la instrucción cuyo número está colocado a continuación de la palabra GOTO para continuar ya después ejecutando por la siguiente instrucción de esta última.

Ejemplo:

```
10 INPUT N%
30 GOTO 50
40 PRINT N%
50 PRINT N%*2
```

En este ejemplo, la sentencia PRINT no se ejecutará nunca. Siempre se salta de la 30 a la 50. De aquí podemos deducir que la instrucción GOTO no tiene sentido usarla de esta manera, porque si yo quisiera que la instrucción 40 no se ejecute nunca, bastaría con no ponerla, con lo que me ahorraría no sólo la 40, sino también la 30.

La instrucción GOTO, pues, tiene sentido utilizarla con instrucciones condicionales (IF), es decir, cuando queremos que el programa «se salte» un grupo determinado de instrucciones sólo en el caso de que se cumpla una condición, mientras que en los demás casos queremos que sí siga ejecutando estas instrucciones.

En el siguiente programa podemos observar cómo se da este caso:

```
10 REM HALLA EL PERIMETRO DE UN TRIANGULO Y DICE SI ES EQUILATERO
20 REM ************************************************************
30 INPUT "LADOS DEL TRIANGULO";A,B,C
40 IF A<>B OR B<>C THEN GOTO 60
50 PRINT "ES EQUILATERO"
60 PRINT "EL PERIMETRO ES";A+B+C
```

También se puede usar el GOTO cuando queremos que un programa vuelva a realizar otra vez lo mismo mientras que no se cumpla una deter-

minada condición de final, como ocurre en el que se ilustra a continuación:

```
10 REM MEDIA DE UNA LISTA DE NUMEROS
20 REM *****************************
30 INPUT "PRIMER NUMERO";N
40 SUMA=SUMA+N :TOTAL=TOTAL+1
50 INPUT "SIGUIENTE NUMERO";N
60 IF N<>-1 THEN GOTO 40
70 PRINT "LA MEDIA ES";SUMA/TOTAL
```

Este programa nos va pidiendo números, que va sumando para después hallar su media, hasta que le introducimos el número −1, que significa que ya se ha acabado la lista de números. En la instrucción 60, si el número tecleado es distinto de −1 va a 40 para sumarlo a la suma acumulada de todos los anteriores y continúa en la siguiente, que es la 50. Cuando introducimos un −1 la condición es falsa y ya no ejecuta la instrucción que va después del THEN(GOTO), sino que sigue por la siguiente, del IF, que es el PRINT, en la que escribe la media de los números.

Hay algunos ordenadores que tienen la posibilidad de poner al final de la instrucción IF la palabra ELSE, y a continuación de ELSE cualquier instrucción o grupo de instrucciones separadas por dos puntos. Cuando el programa ejecuta esta instrucción IF ejecutará la(s) instrucción(es) que haya a continuación del ELSE en el caso de que la condición del IF sea FALSA. Veamos cómo quedaría el primer programa que vimos si pudiéramos usar el ELSE:

```
10 REM PROGRAMA EJEMPLO SENTENCIA IF-THEN-ELSE
20 REM ************************************
30 INPUT "TECLEE UN NUMERO ",N
40 IF(N MOD 2)=0 THEN MENSAJE$="Numero Par" ELSE MENSAJE$="Numero Impar"
60 PRINT MENSAJE$
```

El ELSE no es obligatorio ponerlo y en ordenadores con esta posibilidad también se pueden escribir instrucciones IF, como las que hemos visto hasta ahora (ver fig. 5).

A continuación se ofrece un programa que resume todas las instrucciones que se han visto hasta ahora. Se deja como ejercicio al lector que intente seguir los pasos de lo que hace cada instrucción.

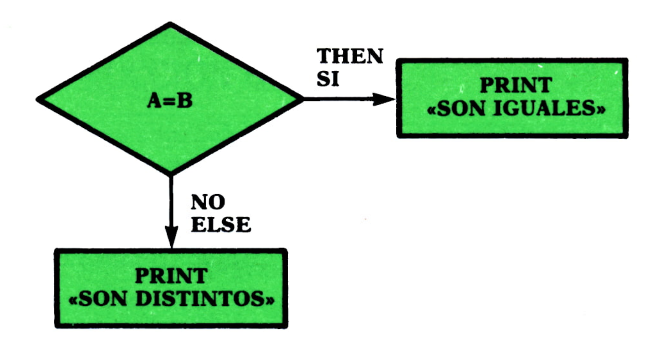


Fig. 5.

```
10 REM PROGRAMA PARA ADIVINAR UN NUMERO COMPRENDIDO ENTRE 1 Y 100
20 REM ******************************************************
30 REM La instruccion 50 hace que el ordenador genere internamente un número al
   azar entre 1 y 100 y lo guarde en la variable X.Se estudiará mas adelante
40 REM Para IBM o compatibles poner 50 RANDOMIZE TIMER :X=INT(RND*100)+1
45 CLS
50 X=INT(RND*100)+1
60 INPUT "TECLEA UN NUMERO";B
70 IF B>X THEN PRINT "te has pasado":GOTO 60
80 IF B<X THEN PRINT "no has llegado":GOTO 60
90 IF B=X THEN PRINT "LO HAS ACERTADO"
100 INPUT "Quieres jugar otra vez(S/N)";J$
110 IF J$="s" OR J$="S" THEN GOTO 50
```

EJERCICIOS:

1. *¿Cuándo se utilizan los operadores lógicos?*
2. *Explicar el operador AND.*
3. *¿Qué ocurre si una instrucción IF no lleva ELSE?*
4. *Haz un programa que te pida una nota y te responda si se trata de un suspenso, aprobado, notable o sobresaliente.*
5. *Haz un programa que te pida una serie de números y te escriba su media geométrica.*
6. *Idea un programa que te pida tres números y te diga cuál es el mayor.*
7. *Haz un programa que ordene tres palabras por orden alfabético.*
8. *Haz un programa que escriba una fila de 50 asteriscos en la pantalla.*
9. *Haz un programa que resuelva una ecuación de segundo grado.*

RESPUESTAS:

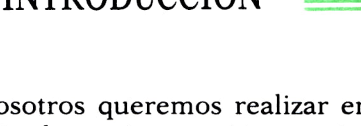
8.

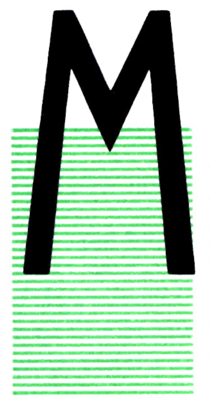
```
10 REM ASTERISCOS
20 REM **********
30 CLS
40 COL=1
50 LOCATE 3,COL:PRINT "*"
60 LET COL=COL+1
70 IF COL<51 THEN GOTO 50
```

9.

```
10 REM ECUACION SEGUNDO GRADO
20 REM **********************
30 INPUT "INTRODUCE COEFICIENTES";A,B,C
40 IF B^2-4*A*C < 0 THEN PRINT "LA ECUACION NO TIENE RAICES REALES":END
45 REM La instrucción END indica al programa que acabe y no siga
50 X1=(-B+((B^2-4*A*C)^(1/2)))/(2*A)
60 X2=(-B-((B^2-4*A*C)^(1/2)))/(2*A)
70 PRINT X1,X2
```

LAS ESTRUCTURAS REPETITIVAS 

INTRODUCCION 



UCHAS operaciones que nosotros queremos realizar en un programa son la repetición de una instrucción o varias un determinado número de veces. Así, por ejemplo, en el programa que vimos en el capítulo anterior que escribía una fila de 50 asteriscos en la pantalla se daba este caso: teníamos que repetir las instrucciones de situarnos en la pantalla en el lugar adecuado y escribir un asterisco 50 veces. Ya vimos entonces una manera de solucionar el problema combinando las instrucciones LET, IF y GOTO. En este capítulo vamos a ver una instrucción específica del BASIC que nos resuelve esta estructura de una manera mas sencilla.

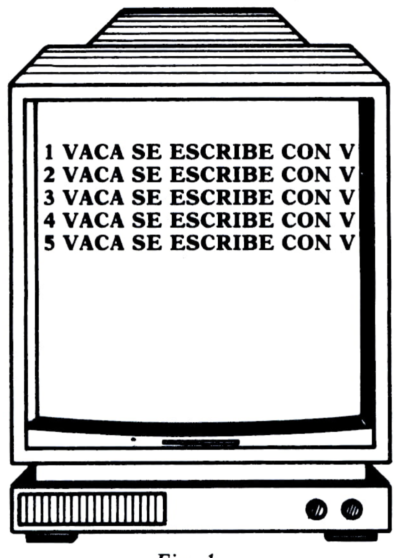


Fig. 1.

Antes de ver la instrucción propiamente dicha, debemos anotar un detalle importante: es muy normal que cuando necesitemos en un programa repetir una instrucción o grupo de instrucciones varias veces nos haga falta también una variable con la que vayamos «contando» las veces que se repite; es decir, una variable que la primera vez que se repiten las instrucciones tenga el valor 1, la segunda el valor 2, y así sucesivamente. En el programa de los asteriscos sería la variable COL que nos va contando las columnas en las que vamos escribiendo cada asterisco. La instrucción que vamos a estudiar en este capítulo también nos proporciona esta «variable-contador».



LA INSTRUCCION FOR

La instrucción FOR tiene en BASIC el siguiente formato:

FOR nombre de variable = valor inicial TO valor final
instrucciones que se quieren repetir

....

....

NEXT nombre de la variable

La variable puede ser cualquiera de tipo numérico y será la que haga las veces de «variable-contador».

Los valores inicial y final serán los valores en los que queramos que empiece y termine la variable. Las instrucciones incluidas entre el FOR y el NEXT se repetirán tantas veces como valores tome la variable desde el inicio hasta el final, ambos inclusive. Estos valores se pueden colocar como constantes, variables (considerándose en este caso el valor que tenga guardado esa variable en ese momento) o expresiones aritméticas (considerándose el resultado de la expresión aritmética). A continuación se pone como ejemplo el programa de escribir la fila de asteriscos utilizando la instrucción FOR:

```
10 REM ASTERISCOS
20 REM *********
30 CLS
40 FOR COL=1 TO 50
50 LOCATE 3,COL:PRINT "*"
60 NEXT COL
```

Veamos otro programa de utilización de FOR:

```
10 REM SUMA DE LOS 1000 PRIMEROS NUMEROS
15 REM ********************************
20 SUMA =0
30 FOR I=1 TO 1000
40 SUMA=SUMA+I
50 NEXT I
60 PRINT "LA SUMA VALE"; SUMA
```

La sentencia 20 inicializa una variable para contener la suma.

La «variable-contador» ha sido llamada I y comienza en 1 y termina en 1000 (instrucción 30)

El funcionamiento del bucle repetitivo es el siguiente:

a) La variable I toma el valor 1.

b) Como I todavía no ha llegado al valor final (1000), se ejecuta el grupo de instrucciones incluidas entre el FOR y el NEXT.

c) La variable SUMA se incrementa en el valor de I. La primera vez se incrementará en 1, la segunda vez en 2, y así sucesivamente.

d) Cuando se llega a NEXT se aumenta en 1 el valor de I y se compara con el valor que hemos puesto como final: si es menor o igual, se volverán a repetir las instrucciones incluidas entre el FOR y el NEXT, valiendo I el número siguiente.

e) Cuando I valga 1000, se hará la instrucción 40 con este valor y se aumentará I en 1. Como I ya valdrá 1001 y es mayor que el valor final, el programa no repetiría más veces y continuaría en la instrucción siguiente del NEXT.



OTRAS CUESTIONES DEL FOR

Vamos a ampliar un poco el estudio de esta importante instrucción repetitiva:

— Puede darse el caso de que nosotros queramos que la variable que nos va contando las veces que se repiten las instrucciones en vez de aumentar de 1 en 1 lo haga de 2 en 2, de 3 en 3 o de cualquier otro valor en otro valor. Para conseguir esto, basta con que al final de la instrucción donde colocamos el FOR pongamos la palabra STEP seguida del número que queremos que sea el aumento del contador. Por ejemplo, si queremos es-

cribir por pantalla todos los números pares del 2 al 100 haríamos el siguiente programa:

```
10 REM NUMEROS PARES DEL 2 AL 100
20 REM ***************************
30 FOR J=2 TO 100 STEP 2
40 PRINT J;
50 NEXT J
```

El incremento del contador que colocamos a continuación de STEP puede ser un número negativo, con lo que el contador contaría hacia atrás. Para este caso, lógicamente el valor inicial debe ser mayor que el final. Veamos uno de los ejemplos anteriores hecho de esta manera:

```
10 REM SUMA DE LOS 1000 PRIMEROS NUMEROS
15 REM *********************************
20 SUMA =0
30 FOR I=1000 TO 1 STEP -1
40 SUMA=SUMA+I
50 NEXT I
60 PRINT "LA SUMA VALE"; SUMA
```

— En ocasiones necesitamos un bucle FOR dentro de otro. Esto está permitido si uno de ellos abarca completamente al otro. Por ejemplo:

```
100 for I=1 to 50
120 for J=1 to 5
    ........
    ........
200 next j
210 next i
```

Este caso sería correcto, ya que lo que significa es que dentro de las instrucciones que hay que repetir desde que I vale 1 hasta que vale 50; para cada valor de I hay que repetir las instrucciones comprendidas entre el FOR J=.... y el next J una vez con J igual a 1, otra con J igual a 2, otra con J igual a 3 y otra con J igual a 4. De esta forma, cada valor de I repetiría 4 veces las instrucciones incluidas en el FOR de J, con lo que estas instrucciones se repetirían 200 veces.

Si hubiéramos puesto:

200 NEXT I
210 NEXT J

la estructura sería incorrecta.

A continuación vemos un programa ejemplo que utiliza bucles **FOR** anidados:

```
10 REM CUADRADO DE PUNTOS
20 REM ******************
25 CLS
30 FOR I=10 TO 20
40 FOR J=40 TO 50
50 LOCATE I,J :PRINT "."
60 NEXT J
70 NEXT I
```

Como para dibujar el cuadrado necesitamos escribir 10 filas de 10 puntos cada una, tendremos que repetir 10 veces la operación de escribir 10 puntos en una fila; por lo que la operación que hay que repetir será a su vez repetir otra operación otro número de veces; es decir, otro bucle **FOR**.

Como resumen de lo visto en este capítulo, se adjuntan a continuación dos programas que usan bucles **FOR**. Se aconseja al lector que trate de seguirlos detenidamente, instrucción a instrucción, para comprobar lo que en ellos se realiza:

```
10 REM FACTORIAL DE UN NUMERO
20 REM **********************
30 INPUT "NUMERO=",N
35 IF N>33 THEN PRINT "EL FACTORIAL DE ESTE NUMERO ES DEMASIADO ";
   "GRANDE PARA MIS CALCULOS":GOTO 30
40 F=1
50 FOR I=N TO 2 STEP -1
60 F=F*I
70 NEXT I
80 PRINT "EL FACTORIAL ES";F
```

```
10 REM   **************************************************************
20 REM   **      PROGRAMA PARA HALLAR LA TABLA DE NUMEROS PRIMOS      **
30 REM   **      ENTRE DOS NUMEROS DADOS                              **
40 REM   **                                                           **
70 REM   **************************************************************
```

```
80 REM LA INSTRUCCION 90 PARA COMMODORE ES :90 PRINT CHR$(147)
90 CLS
100 INPUT "INTRODUCE LOS NUMEROS ENTRE LOS QUE QUIERES CALCULAR LA TABLA";A,B
110 IF A<3 THEN PRINT 1;2;:LET A=3
120 IF (A/2)=INT(A/2) THEN LET A=A+1
130 FOR K=A TO B STEP 2
140 FOR C=3 TO INT(SQR(K))
150 IF K/C = INT(K/C) THEN GOTO 180
160 NEXT C
170 PRINT K;
180 NEXT K
190 REM La expresion  INT(...) significa la parte entera de lo
200 REM que hay entre paréntesis.(sin los decimales).Se verá mas adelante
```

Hay veces que en un programa queremos repetir algo y, sin embargo, no podemos utilizar la instrucción FOR. Esto ocurre cuando no sabemos «a priori» las veces que se tiene que repetir, sino que dependiendo de las circustancias, cada vez que ejecutemos el programa, se va a repetir un número distinto de veces. Este era el caso, por ejemplo, del programa que veíamos en el capítulo anterior para calcular la media de una serie de números. Cada vez que lo ejecutábamos, el número de números de los que íbamos a hallar la media era distinto, y por tanto, las instrucciones de pedir un número y sumarlo a la suma acumulada de los anteriores se repetirán cada vez un número distinto de veces. Si quisiéramos hacer esto con un FOR, ¿qué valor final íbamos a poner para que se terminase la repetición si no sabemos cuántas veces se repetirá cada vez? En este caso, la única solución es la que se dio en el capítulo anterior, o bien la utilización de la instrucción WHILE-WEND en aquellos ordenadores que la posean. Esta instrucción, que es más propia de los métodos de programación estructurada, será estudiada con profundidad en el próximo número que se dedicará a BASIC avanzado, ya que se trata de una instrucción no estándar del BASIC.

EJERCICIOS:

1. Haz un programa que construya la tabla de multiplicar de un número.

2. Haz un programa que calcule: 1+ 1/3 + 1/5 + 1/7 + ... + 1/N, siendo N un número impar que te pida el programa previamente.

3. Haz un programa que te haga el siguiente dibujo (utilizando la instrucción FOR):



4. Idea un programa que te pida por pantalla 10 números y te escriba el mayor de ellos.

5. Haz un programa que dibuje en pantalla un rectángulo de puntos.

RESPUESTAS:

1.

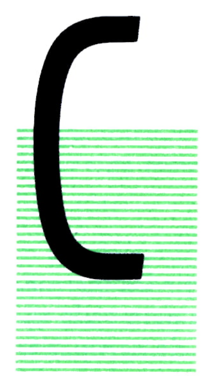
```
10 REM TABLA DE MULTIPLICAR
20 REM ********************
30 INPUT"NUMERO DEL QUE QUIERES SABER LA TABLA=",N
40 FOR I=1 TO 9
50 PRINT N;"x";I;"=";N*I
60 NEXT I
```

4.

```
10 REM MAYOR DE 10 NUMEROS
20 REM ******************
25 INPUT "INTRODUCE PRIMER NUMERO=",MAX
30 FOR I=2 TO 10
40 INPUT "INTRODUCE SIGUIENTE NUMERO=",N
50 IF N>MAX THEN MAX=N
60 NEXT I
70 PRINT "EL MAYOR ES ";MAX
```

LA FORMA DE ESTRUCTURAR LOS DATOS 10

INTRODUCCION



CUANDO estudiamos las variables veíamos que éstas eran unas casillas de memoria donde guardábamos un valor numérico o alfanumérico. En cada variable se guardaba un único valor. De esta forma, cuando en un programa necesitábamos guardar una serie de valores distintos debíamos utilizar tantas variables como valores necesitásemos y luego referenciarlas en el programa a cada una por su nombre.

En este capítulo vamos a ver una estructura que nos introduce una mejora a este respecto: vamos a poder agrupar un conjunto de variables (las que queramos), con un mismo nombre y nos referiremos a ellas por el número de orden que ocupe cada una.

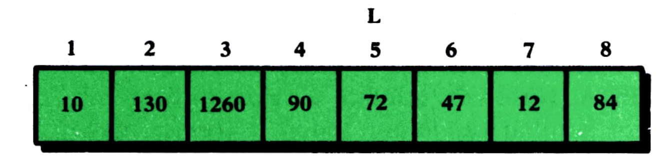


Fig. 1.

En la figura vemos que hemos agrupado 8 variables bajo el nombre de L y distinguiremos una de otra por su número.

Al igual que siempre, si las variables fueran alfanuméricas, el nombre deberá terminar en $.

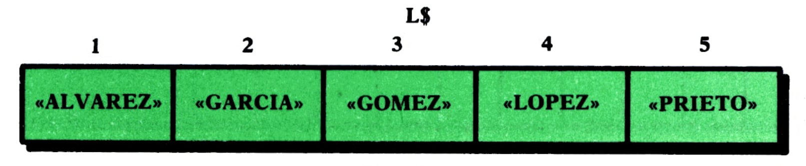


Fig. 2.

Este tipo de variables nos supone grandes ventajas, ya que cuando queramos recorrer un conjunto de variables para hacer con todas ellas lo mismo, podemos usar una instrucción FOR cuyo contador será precisamente el número de la variable, con lo que sólo tendremos que escribir lo que queremos hacer con todas una vez dentro del FOR.

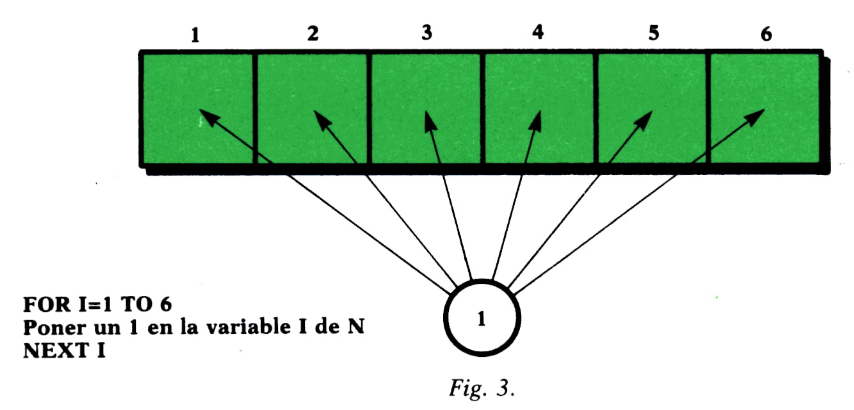


Fig. 3.

Si no utilizásemos esta estructura, tendríamos que repetir las instrucciones de lo que queramos hacer con las variables una vez para cada variable, ya que al tener nombres distintos, no habría forma de agruparlas en un FOR.



SU DEFINICION EN BASIC

Este tipo de estructura suele recibir muchos y distintos nombres. Así, entre los más comunes tenemos: VARIABLE DIMENSIONADA, VECTOR, MATRIZ o LISTA. Nosotros, aunque a veces utilizaremos cualquiera de ellos para irnos familiarizando con todos, usaremos mayoritariamente en el texto el de LISTA, que quizá es el que se acerca más a la idea intuitiva de lo que es (una lista de variables).

Las listas en un programa se utilizan de la manera siguiente:

— Antes de la primera instrucción en la que vayamos a utilizar la lista debemos escribir la instrucción DIM, cuyo formato es:

N. de instr. DIM nombre de la variable (n. de variables de que consta).

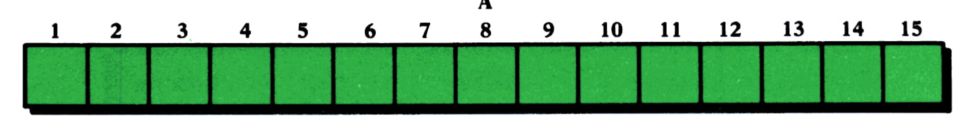


Fig. 4.

Si en un programa fuésemos a utilizar la lista de la figura, deberíamos escribir la instrucción:

DIM A(15)

Esta instrucción DIM sirve para indicarle al programa que vamos a usar una lista que vamos a llamar de la manera que le indiquemos y que va a constar de las variables que pongamos. Hay que ponerla antes de utilizar la lista por primera vez y el programa la DEBE EJECUTAR UNA UNICA VEZ, por lo que no debe encontrarse ni dentro de un bucle FOR ni en un grupo de instrucciones que se ejecuten más de una vez, porque se vuelva a ellas a través de una instrucción GOTO.

— Para utilizar las variables de la lista en el programa debemos poner el nombre de la lista y entre paréntesis el número de la variable a la que nos estemos refiriendo. Así, si en el programa en el que tengo declarada la lista anterior quiero guardar un 100, en la variable número 7 tendré que poner una instrucción:

LET A(7)=100

Las variables de la lista se comportan como variables normales de manera individual y con ellas podemos hacer todo lo que podíamos hacer con las variables que hemos visto hasta ahora usándolas individualmente.

Conviene resaltar también que en BASIC NO existe ninguna instrucción que nos permita utilizar todas las variables de una lista de manera conjunta, de forma que SIEMPRE tendremos que manejar las variables de una lista UNA A UNA. Esto no quita que las usemos una a una, pero dentro de un bucle FOR, con lo que nosotros sólo tendríamos que escribir lo que queremos hacer con todas ellas (siempre que sea lo mismo) una vez, aunque en la ejecución el programa lo repetirá una vez para cada una de ellas, tal y como ocurre con la ejecución de una instrucción FOR. En el siguiente apartado vamos a ver distintos ejemplos de este caso.

A continuación veamos un primer y sencillo ejemplo del manejo de listas:

```
10 REM CALCULO DE LAS HORAS DE TRABAJO SEMANALES
20 REM ****************************************
25 REM En cada variable de la lista S guardamos las horas
27 REM de trabajo de cada dia de la semana
30 DIM S(7)
40 REM DE LUNES A VIERNES SON 7 HORAS DIARIAS SIEMPRE
50 FOR I=1 TO 5
60 LET S(I)=7
70 NEXT I
80 INPUT "HORAS DE TRABAJO DEL SABADO";S(6)
```

```
90 INPUT "HORAS DE TRABAJO DEL DOMINGO";S(7)
100 REM HORAS TOTALES DE TRABAJO A LA SEMANA
110 H=0
120 FOR I=1 TO 7
130 H=H+S(I)
140 NEXT I
150 PRINT "LAS HORAS DE TRABAJO A LA SEMANA SON ";H
```

En la figura podemos observar lo que haría este programa:

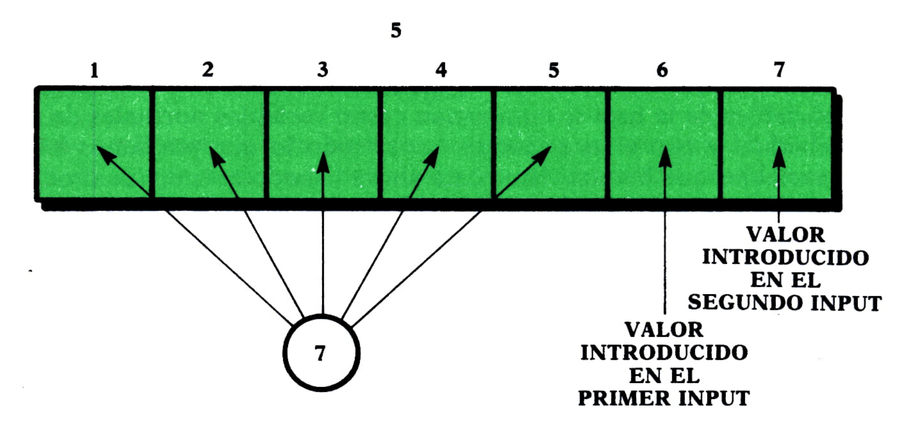


Fig. 5.



OPERACIONES TIPICAS CON LISTAS

Vamos a estudiar en este apartado una serie de operaciones que podemos necesitar realizar con una lista en numerosos programas.

Para todos ellos vamos a suponer que tenemos declarada la lista:

DIM A(5)

aunque se pueden generalizar para cualquier lista.

1. Escribir por pantalla los valores guardados en las variables de una lista:

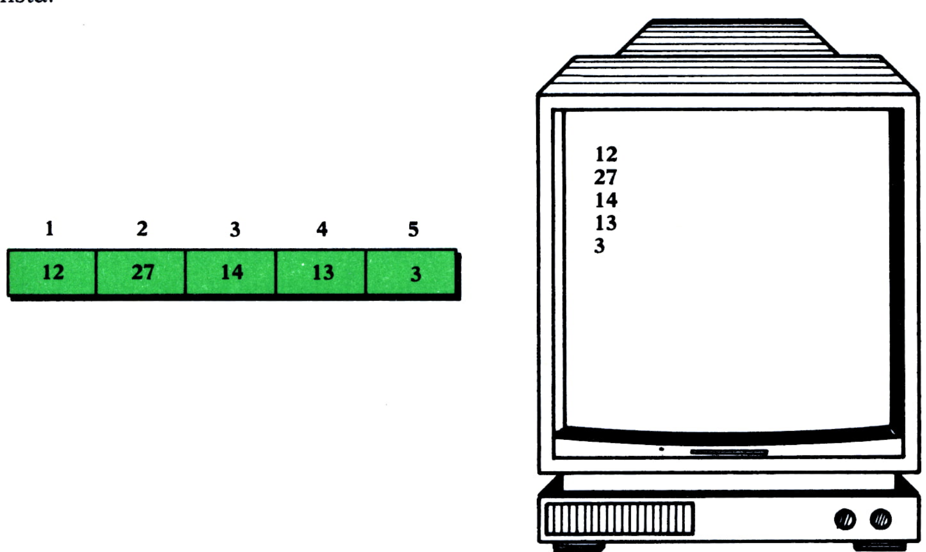


Fig. 6.

```
10 REM ESCRIBIR
40 FOR I=1 TO 5
50 PRINT A(I);
60 NEXT I
```

2. Guardar en todas las variables de la lista un mismo valor:

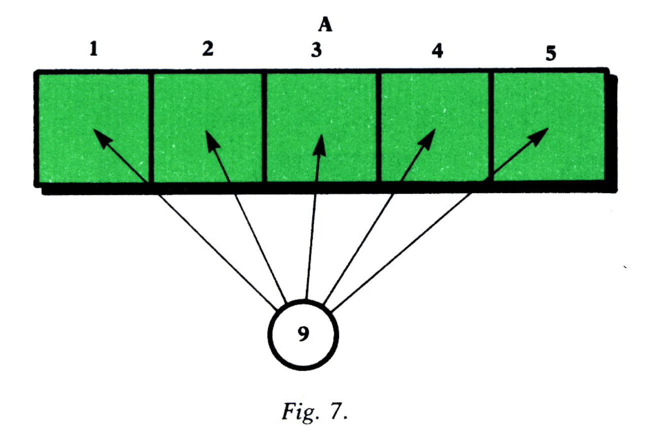


Fig. 7.

```
10 REM GUARDAR
40 FOR I=1 TO 5
50 LET A(I)=9
60 NEXT I
```

3. Ir pidiendo valores por pantalla e ir introduciéndolos en las variables de la lista en orden:

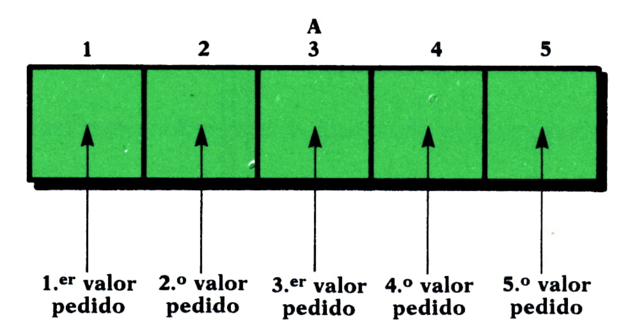
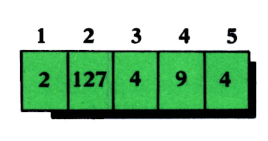
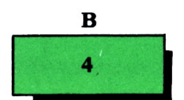


Fig. 8.

```
10 REM PEDIR
40 FOR I=1 TO 5
45 PRINT "VALOR NUMERO";I;
50 INPUT A(I)
60 NEXT I
```

4. Buscar en qué variable de la lista tenemos guardado un cierto valor:





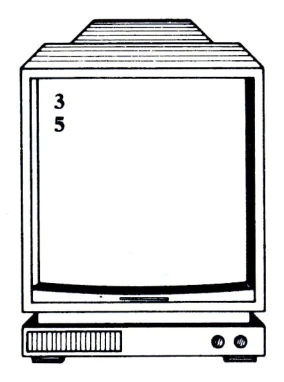


Fig. 9.

```
10 REM BUSCAR
20 REM ******
30 FOR I=1 TO 5
40 IF B=A(I) THEN PRINT "EL ELEMENTO BUSCADO ESTA EN ";I
50 NEXT I
```

Todos estos programas tienen en común que recorren las variables de la lista con una instrucción **FOR** y realizan lo mismo para cada una de ellas utilizando la «variable-contador» del **FOR** para llamar cada vez a cada una de las distintas variables de la lista y de esta forma sólo tenemos que escribir una vez lo que queremos hacer con cada una de ellas.

El siguiente programa es un ejemplo de aplicación de estos casos vistos anteriormente:

```
10 REM CONTROL DE STOCK DE ARTICULOS EN UN ALMACEN
20 REM ******************************************
30 DIM A(5)
40 REM SOLICITUD DEL STOCK EN ALMACEN DE CADA ARTICULO
50 FOR I=1 TO 5
60 PRINT "STOCK INICIAL DEL ARTICULO";I
70 INPUT A(I)
80 NEXT I
85 REM STOCK VENDIDO EN UN ESPACIO DE TIEMPO
90 FOR I=1 TO 5
100 PRINT "UNIDADES VENDIDAS DEL ARTICULO";I
110 INPUT UNI
120 LET A(I)=A(I)-UNI
130 NEXT I
140 REM BUSQUEDA DE LOS ARTICULOS CUYO STOCK ES MENOR QUE 10
150 FOR I=1 TO 5
160 IF A(I)<10 THEN PRINT "HAY QUE COMPRAR MAS UNIDADES DEL ARTICULO";I
170 NEXT I
180 REM IMPRESION POR PANTALLA DEL STOCK DE CADA ARTICULO
190 FOR I=1 TO 5
200 PRINT "EL STOCK DEL ARTICULO";I;"ES";A(I)
210 NEXT I
```



LAS TABLAS

La instrucción DIM del BASIC también nos permite definir lo que llamamos TABLAS. Estas tablas responden a una estructura como la que se ilustra en la figura adjunta en la que cada casilla es una variable.

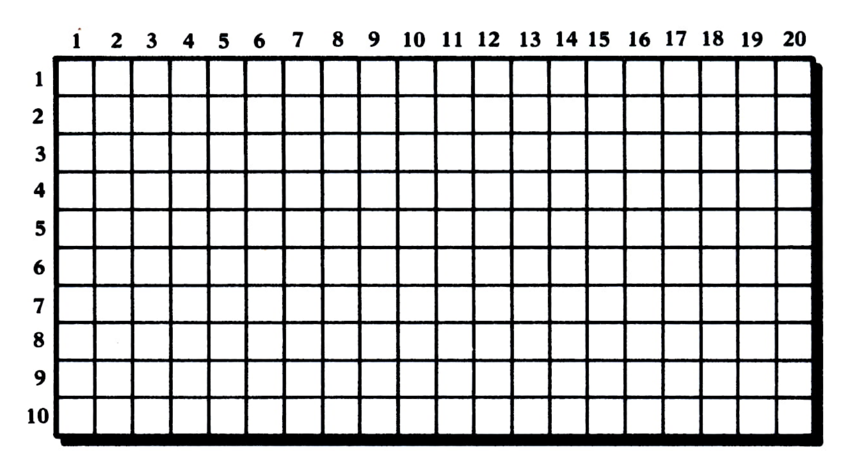


Fig. 10.

Las tablas las definiríamos poniendo, por ejemplo:

DIM T(10,20)

que significaría que la tabla va a tener 10 filas y 20 columnas.

Al igual que las listas, las tablas no podemos utilizarlas de manera conjunta, sino que debemos usarlas variable a variable. Para referenciar en un programa la variable de la fila 7, columna 12 de la tabla anterior, deberíamos poner, por ejemplo:

LET T(7,12)=5

Este sería el caso en que quisiésemos guardar un 5 en dicha variable.

A continuación se ofrece un programa para escribir los valores guardados en todas las variables de la tabla:

```
10 FOR I=1 TO 10
20 FOR J=1 TO 20
30 PRINT T(I,J);
40 NEXT J
45 PRINT
50 NEXT I
```

Puede observar el lector que es la misma idea que para el caso de la lista, con la diferencia de que en este caso necesitamos utilizar bucles FOR anidados, ya que para cada fila de la tabla hay que recorrer todas las columnas, es decir, hay que hacer una operación que consiste en la repetición de otra. Se deja como ejercicio al lector la realización para tablas de

las otras operaciones típicas estudiadas para listas, así como cualquier otra que pueda ocurrirse.

Con la instrucción **DIM** también podemos definir tablas de tres dimensiones o más, siendo la técnica para hacerlo la misma. Una tabla de tres dimensiones la definiríamos:

DIM T(10,25,50)

y a una variable de esa tabla habría que llamarla poniendo:

T(3,14,20), por ejemplo, si queremos llamar a la variable (3,14,20).

Su manejo sería, por tanto, igual que para el caso de una o dos dimensiones.

A continuación vemos tres programas ejemplo en los que resulta imprescindible el empleo de variables dimensionadas:

```
10 REM CONTABILIDAD DE FACTURAS
20 REM ************************
25 CLS
30 INPUT "NUMERO DE FACTURAS";N
40 DIM T(N,2)
50 FOR I=1 TO N
55 CLS:LOCATE 10,25
60 PRINT "FACTURA";I
70 INPUT "COBRO O PAGO (P/C)";F$
75 REM T(I,2)=1 si es un pago y =2 si es un cobro
80 IF F$="P" THEN T(I,2)=1 ELSE T(I,2)=2
90 INPUT "IMPORTE DE LA FACTURA";T(I,1)
100 NEXT I
110 FOR I=1 TO N
120 IF T(I,2)=1 THEN PAG=PAG+T(I,1)
130 IF T(I,2)=2 THEN COB=COB+T(I,1)
140 NEXT I
145 CLS :LOCATE 10,1
150 PRINT "PAGOS TOTALES REALIZADOS=";PAG
160 PRINT
170 PRINT "COBROS TOTALES REALIZADOS=";COB
```

```
10 REM RESERVA DE BILLETES DE FERROCARRIL
20 REM **********************************
30 DIM T(49): DIM P(49)
40 REM Hay 49 plazas.La lista T tiene el número de tren de cada plaza.
50 REM P tiene un 0 si la plaza está vacía y un 1 si está ocupada.
60 REM INICIALIZACION DE LOS Nº DE TREN DE CADA PLAZA
70 FOR I=1 TO 49
80 LET T(I)=I\10 + 1
```

```
90 NEXT I
100 REM RESERVA DE UNA PLAZA
105 CLS
110 INPUT "NUMERO DE TREN EN QUE SE QUIERE RESERVAR";NT
111 CLS
115 IF NT=0 THEN GOTO 160
120 FOR I=1 TO 49
130 IF T(I)=NT AND P(I)=0 THEN P(I)=1:PRINT "PLAZA";I MOD 10 +1 ;
   "RESERVADA":GOTO 110
140 NEXT I
150 PRINT "NO HAY PLAZAS EN ESE TREN"
155 GOTO 110
160 REM IMPRESION DE LAS PLAZAS RESERVADAS
165 CLS:LOCATE 2,20:PRINT "PLAZAS RESERVADAS":LOCATE 4,1
170 FOR I=1 TO 49
180 IF P(I)=1 THEN PRINT "PLAZA";I MOD 10 + 1  ;"DEL TREN";I\10 +1
190 NEXT I
```

```
10 REM INTRODUCCION DE PALABRAS POR TECLADO E IMPRESION DE ESTAS
20 REM *******************************************************
30 INPUT "NUMERO DE PALABRAS";NP
40 DIM P$(NP)
50 FOR I=1 TO NP
60 INPUT P$(I)
70 NEXT I
80 FOR I=1 TO NP
85 LPRINT P$(I);" " ;
90 NEXT I
```



VARIACIONES PARA EL SPECTRUM

El SPECTRUM tiene algunas restricciones en lo que al manejo de listas se refiere, que a continuación se relatan:

— Los nombres de todas las variables dimensionadas deben constar de una sola letra (si son alfanuméricas una letra y el símbolo $).

— En la instrucción DIM, para variables dimensionadas alfanuméricas se debe poner, antes de cerrar el paréntesis, un número que indique el máximo de letras que pueden tener las palabras que vamos a guardar en ella. Por ejemplo:

La declaración 20 DIM A$(5) en el SPECTRUM debería ponerse:

20 DIM A$(5,20) si suponemos que las cinco palabras que se van a guardar en las cinco variables de la lista van a tener un máximo de 20 letras.

— El nombre de las variables-contador de las instrucciones **FOR** debe constar de una sola letra, al igual que las variables alfanuméricas sencillas (una letra y el **$**).

EJERCICIOS:

1. Realiza un programa que te pida las letras de un determinado texto (una a una y con los espacios de separación correspondientes) y a continuación te escriba el texto al revés. El programa deberá pedir previamente el número de letras de que consta el texto.

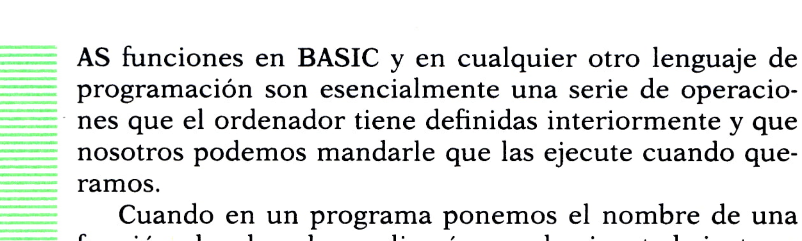
2. Haz un programa que sume dos números enteros de 100 cifras.

3. Modifica el programa que se expone en el capítulo sobre el control de stock de una serie de artículos en un almacén para que por cada artículo se tenga también el precio unitario de éste y al final junto con las existencias finales de cada uno se obtenga también el precio total de dichas existencias.

LAS FUNCIONES 11



INTRODUCCION



LAS funciones en **BASIC** y en cualquier otro lenguaje de programación son esencialmente una serie de operaciones que el ordenador tiene definidas interiormente y que nosotros podemos mandarle que las ejecute cuando queramos.

Cuando en un programa ponemos el nombre de una función el ordenador realizará, cuando ejecute la instrucción donde se encuentre, las operaciones correspondientes a ese nombre.

Las funciones, sin embargo, tienen una propiedad fundamental: como fruto de esa serie de operaciones que se realizan, por cada función el ordenador siempre terminará **OBTENIENDO UN VALOR** que podrá ser numérico o alfanumérico. De esta manera, cuando en un programa utilicemos una función, siempre debemos indicar que queremos hacer con el valor resultado de ésta: asignarlo a una variable, escribirlo por pantalla, utilizarlo como valor inicial o final de una instrucción **FOR**, etc. Por eso, una función en un programa siempre se debe colocar en un lugar destinado a valores: después del igual de una instrucción **LET**, a continuación de una instrucción **PRINT**, como operando de una expresión aritmética (su resultado), etc.



COMO SE ESCRIBE UNA FUNCION

Una función en un programa se escribe poniendo su nombre y a continuación, entre paréntesis, una serie de datos (cuando los tiene), que son datos a partir de los cuales la función obtendrá el **VALOR RESULTADO**. Estos datos suelen recibir el nombre de argumentos o datos de entrada.



Fig. 1.

En este capítulo vamos a ver todas las funciones más importantes que existen en BASIC y lo que hace cada una de ellas.

Al igual que en el caso de las variables, cuando una función obtiene como resultado un valor alfanumérico, su nombre terminará con el símbolo $ y cuando obtenga un valor numérico, no llevará este símbolo.



LAS FUNCIONES NUMERICAS

Las funciones numéricas son aquellas cuyo resultado es un valor numérico. Veamos las más importantes:

ABS(N). Obtiene el valor absoluto del dato colocado como argumento (N).

Recordad que los datos que pongamos entre paréntesis como argumentos de las funciones, en un programa podrán ser constantes, variables o expresiones aritméticas y para estos dos últimos casos la función tomará el valor que haya en ese momento en la variable o el valor resultado de la expresión aritmética.

Así, veamos, como ejemplo, cómo podríamos utilizar la primera función estudiada:

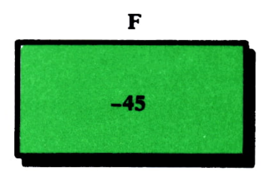


Fig. 2.

Si tenemos el caso de la figura y en un programa se ejecuta la instrucción:

LET A=ABS(F)

nos quedará:

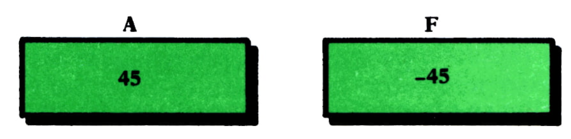


Fig. 3.

ATN(N). Obtiene el arco cuya tangente es N.

CINT(N). Obtiene la parte entera de N redondeando la parte decimal.

COS(N). Obtiene el coseno de N.

EXP(N). Obtiene el resultado de elevar el número «e» a N.

INT(N). Obtiene la parte entera de N truncando la parte decimal.

LOG(N). Obtiene el logaritmo natural de N.

RND. Obtiene un número elegido por el ordenador al azar entre 0 y 1. Hay algunos ordenadores que exigen que esta función lleve un argumento entre paréntesis. Se aconseja, por ello, consultar el manual antes de utilizarla.

Vamos a ver con un poco más de detenimiento esta función porque resulta muy interesante en numerosos programas. Cuando nosotros ponemos:

LET A=RND

la función obtendrá como resultado un número que elija al azar entre 0 y 1 (decimal lógicamente), y lo guardará en la variable A. Pero en muchos programas nosotros no queremos elegir un número al azar entre 0 y 1, sino entre 2 números cualesquiera, que podemos generalizar poniendo N y M. Para conseguir esto, bastará con poner:

LET A=(RND*(M-N+1))+N

Si queremos que nos obtenga un número al azar entre N y M y que además este número sea entero siempre, bastaría con utilizar la función INT vista anteriormente y poner:

LET A=INT(RND*(M-N+1))+N

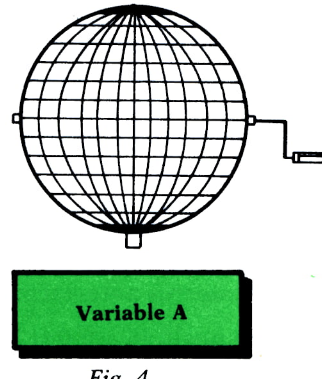


Fig. 4.

Para que el programa obtenga un número entero al azar entre 1 y 100 (N=1 y M=100) pondríamos:

LET A=INT(RND*100)+1

A continuación se ofrecen dos programas que simulan un sorteo de lotería primitiva obteniendo seis números al azar entre 1 y 49. El segundo evita que puedan salir números repetidos marcando los que salen: cuando sale un número pone la variable de la lista que tiene el mismo número a 1.

```
10 REM LOTERIA PRIMITIVA
20 REM ****************
30 RANDOMIZE TIMER 'SOLO PARA IBM (MS-DOS)
40 FOR I=1 TO 6
50 LET N=INT(RND*49)+1
60 PRINT N
70 NEXT I
```

```
10 REM LOTERIA PRIMITIVA
20 REM ****************
30 DIM M(49)
40 RANDOMIZE TIMER 'SOLO PARA IBM (MS-DOS)
50 FOR I=1 TO 6
60 LET N=INT(RND*49)+1
70 IF M(N)=1 THEN GOTO 60
80 M(N)=1
90 PRINT N
100 NEXT I
```

El lector puede ejecutar el programa obteniendo cada vez una combinación de seis números distinta y ...¡suerte!

Hay una instrucción que está estrechamente relacionada con la función RND: la instrucción RANDOMIZE. Esta instrucción se utiliza para que el ordenador inicialice el mecanismo que tiene para generar números al azar. Se debe poner en TODOS los programas en los que se utilice la función RND, al principio de éstos, aunque UNICAMENTE es obligatorio utilizarla para ordenadores que tengan sistema operativo MS-DOS(PC's o compatibles). En estos ordenadores se debe poner al principio de los programas en los que se use la RND la instrucción:

RANDOMIZE TIMER

ya que si no se pone siempre se obtendrán los mismos números de la función RND. En los programas puestos anteriormente, así se indica este caso.

SGN(N). Obtiene un 1 si N es positivo, un 0 si N es 0 y un –1 si N es negativo.

SIN(N). Obtiene el seno de N.
SQR(N). Obtiene la raíz cuadrada positiva de N.
TAN(N). Obtiene la tangente de N. Para estas funciones trigonométricas se aconseja consultar antes el manual del ordenador para ver si el argumento debe ir en grados o radianes.

A continuación se ofrece un programa que sirve de ejemplo de utilización de las funciones estudiadas hasta ahora.

```
10 REM EJEMPLO-FUNCIONES NUMERICAS
20 REM ***************************
25 CLS
30 INPUT "INTRODUCE UN NUMERO";N
40 PRINT "LA RAIZ CUADRADA DE";N;"ES";SQR(N)
50 PRINT "EL LOGARITMO DE     ";N;"ES";LOG(N)
60 PRINT "LA PARTE ENTERA DE ";N;"ES"; INT(N)
```

ASC(N$). Obtiene el código ASCII del caracter inicial de N$. El código ASCII es una numeración interna que tienen todos los caracteres de nuestro ordenador. Esta función nos la proporciona $. Recuerda que N$ representa cualquier constante alfanumérica, variable o expresión.
INSTR(N,A$,B$). Busca a partir de la letra N de la palabra A$ el carácter inicial de B$. Obtiene como resultado el número de la posición que ocupa este carácter en A$. Si no se pone la N, se toma como si fuera un 1. Si no encuentra el carácter, devuelve un cero.

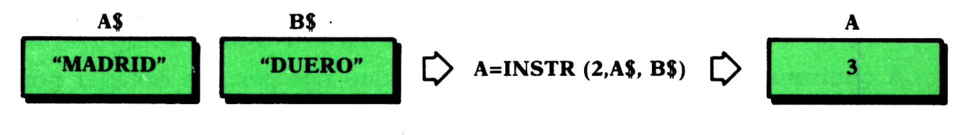
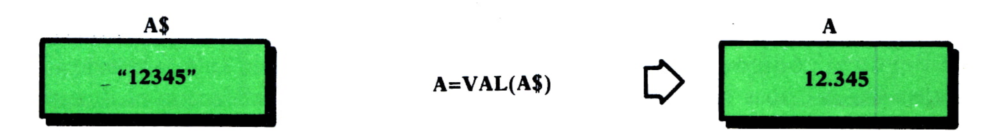


Fig. 5.

LEN(A$). Obtiene como resultado la longitud (número de caracteres) de A$.
VAL(A$). Si A$ está compuesta por números exclusivamente, obtiene como resultado el número correspondiente de considerar los números de A$ como una cifra numérica. Si A$ está compuesta por algún carácter distinto de los números, obtendrá como resultado un 0.



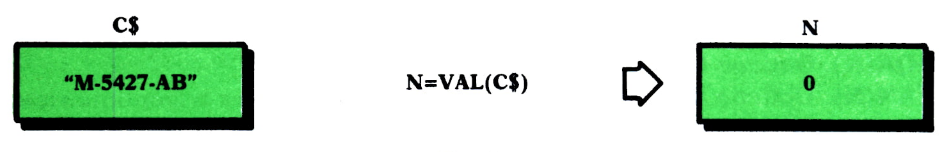


Fig. 6.

CSRLIN. Obtiene la línea donde terminó de escribir el cursor en pantalla la última vez.
POS. Obtiene la columna donde terminó de escribir el cursor en pantalla la última vez.



FUNCIONES ALFANUMERICAS

Las funciones alfanuméricas serán aquellas que obtengan como resultado un valor alfanumérico. Por tanto, deberán ser asignadas siempre a variables alfanuméricas. Las más importantes son:

CHR$(N). Obtiene el carácter alfanumérico cuyo código ASCII es N.

Hay 256 códigos ASCII numerados del 0 al 255. Todos los caracteres del teclado tienen asignado un código ASCII, pero además hay códigos ASCII asignados a caracteres especiales (de tipo gráfico) que no tienen ninguna tecla asociada y códigos ASCII que corresponden al borrado de pantalla, sonido y algunos otros de este estilo. Esta función CHR$ nos permite escribir en pantalla los caracteres gráficos que no tienen tecla asignada o realizar las funciones de borrado de pantalla, música y demás poniendo:

PRINT CHR$ (código del carácter que queremos)

A continuación se ofrece un programa que nos permite obtener por pantalla todos los códigos ASCII con el carácter que corresponde a cada uno.

```
10 REM CODIGOS ASCII
20 REM ************
30 CLS
40 FOR I=0 TO 255
50 PRINT I;"-->"; CHR$(I)
60 NEXT I
```

Al ejecutar el programa anterior hay que tener en cuenta que cuando se llegue al código correspondiente al borrado de pantalla se nos borrarán todos los escritos hasta ahora. Se deja como ejercicio al lector el descubrimiento de cuál es el código que corresponde a borrar la pantalla y la

modificación del programa para que no ejecute la instrucción PRINT con este código y no nos borre los caracteres anteriores.

Como en la pantalla no caben los 256 caracteres, al ejecutar el programa habrá que ir parando la visualización por pantalla con las teclas correspondientes, para ir así viendo por partes lo que nos vaya escribiendo. También existe la posibilidad de ejecutar el programa por partes poniendo cada vez en la instrucción FOR un rango de números, de tal manera que quepan todos en pantalla. Ejecutando el programa cada vez con un grupo de números distintos en el FOR terminaríamos igualmente viendo todos los caracteres.

Obsérvese que en este programa lo que se hace es mandar escribir el resultado de la función CHR$ que será, por ser función alfanumérica, un valor de este tipo, es decir, un carácter.

Estos caracteres especiales que no tienen ninguna tecla asociada se comportan en un programa como aquéllos que sí tienen tecla, es decir, que podemos guardarlos en una variable alfanumérica poniendo, por ejemplo:

LET C$=CHR$(250)

y también podemos guardarlos en variables formando palabras con otros. Así, también podríamos poner:

LET C$=CHR$(250)+«a»+CHR$(199)

lo que nos guardaría en la variable C$ una palabra formada por tres caracteres: el primero, aquél cuyo código es 250, el segundo, la letra a minúscula, y el tercero, aquél cuyo código es 199.

LEFT$(A$,N). Obtiene los N primeros caracteres de A$ comenzando a contar por el primero de su izquierda.

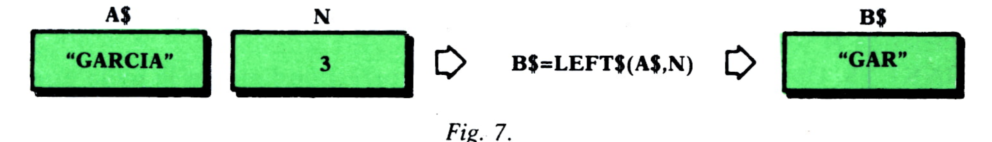


Fig. 7.

MID$(A$,N,M). Obtiene M letras consecutivas de A$ contando a partir de N.

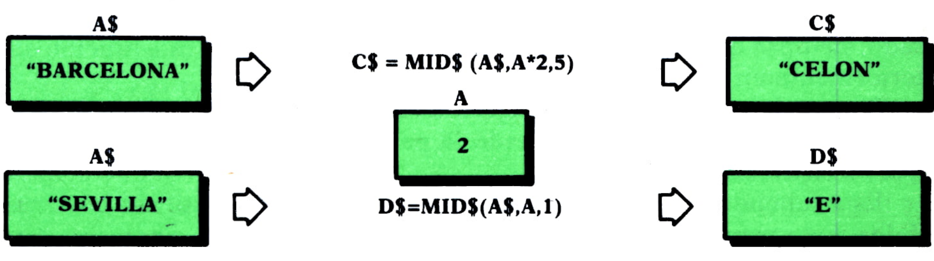


Fig. 8.

Si no se pone la longitud de la palabra que se quiere obtener (es decir, M) o hay menos caracteres a la derecha del carácter número N de A$, la función obtiene todos los caracteres que haya a la derecha del N-ésimo.

Si M es cero o N mayor que la longitud de A$, la función obtiene como resultado la palabra vacía; que en BASIC se expresa: «».

RIGHT$(A$,N). Obtiene los N caracteres consecutivos de A$, comenzando a contar por el primero de su derecha.

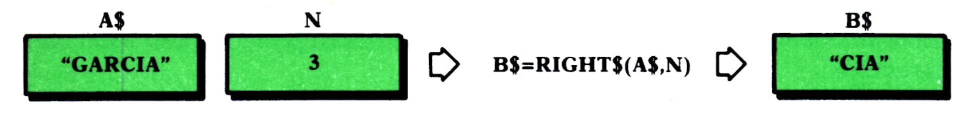


Fig. 9.

SPACE$(N). Obtiene como resultado un valor alfanumérico compuesto por N espacios en blanco.

Si ponemos: PRINT SPACE$(5); A nos dejará cinco espacios en blanco en la línea en la que esté escribiendo y a continuación nos escribirá el valor que haya guardado en la variable A.

Si pusiéramos: LET A$=SPACE$(11) nos guardaría en A$: « » (11 espacios en blanco).

STRING$(N,M). Obtiene una palabra formada por N caracteres iguales. Estos caracteres serán aquéllos cuyo código ASCII sea M.

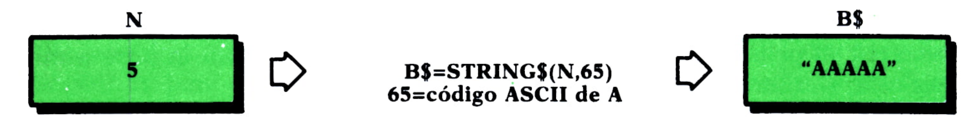


Fig. 10.

STRING$(N,A$). Obtiene una palabra formada por la letra inicial de A$ repetida N veces.

INKEY$. Obtiene como resultado la letra correspondiente a la tecla que se haya pulsado desde que comenzó la ejecución del programa. Si se han pulsado varias, la primera vez que se utilice la función INKEY$ obtendrá la primera letra que se pulsó, la segunda vez la segunda, y así sucesivamente. Es decir, obtiene la letra más antigua de todas las que se hayan pulsado, siempre que no haya sido ya dada como resultado de esta función anteriormente en el programa.

Si cuando en el programa se ejecuta esta función no se ha pulsado ninguna tecla, obtiene como resultado la palabra vacía, que ya hemos visto que se representa por «».

Esta función es muy importante, ya que nos permite utilizar las letras del teclado para conseguir que durante la ejecución de un programa éste haga diversas cosas según las teclas que pulsemos.

Veamos, por ejemplo, un par de instrucciones que nos permiten que el programa se quede sin hacer nada hasta que nosotros pulsemos una tecla:

```
10 A$=INKEY$
20 IF A$="" THEN GOTO 10
```

A continuación de estas instrucciones podríamos poner otras de la forma:

IF A$=«F» THEN ......
IF A$=«G» THEN ....
....

Esto significaría que según la tecla que se haya pulsado, el programa haría lo que nosotros pongamos a continuación de los **THEN.** Si suprimimos la instrucción:

IF A$=«» THEN GOTO 10

conseguiríamos que el programa no se parase, sino que si se han pulsado las teclas indicadas en los IF, ejecutase lo que hemos puesto y si no continuase ejecutando las instrucciones que vengan después.

A continuación observamos dos programas en los que se usa esta función:

```
10 REM PROGRMA PARA DIBUJAR EN PANATALLA MEDIANTE 4 TECLAS
20 REM ***************************************************
30 CLS
40 F=5:C=10
50  LOCATE F,C :PRINT "*"
60 A$=INKEY$
70 IF A$="" THEN GOTO 60
80 REM TECLAS PARA MOVERSE: U,J,H,N
85 REM ******************************
90 REM F PARA TERMINAR
100 IF A$="U" THEN IF F>1 THEN F=F-1
110 IF A$="H" THEN IF C>1 THEN C=C-1
120 IF A$="J" THEN C=C+1
130 IF A$="N" THEN F=F+1
140 IF A$="F" THEN END
150 LOCATE F,C :PRINT "*"
160 GOTO 60
```

```
10 REM VISUALIZACION DE UN CRONOMETRO DIGITAL
20 REM ************************************
30 CLS
40 REM RETARDO DE UN SEGUNDO
50 FOR I=1 TO 700:NEXT I
55 REM Ajustar en cada ordenador el valor final del FOR
56 REM hasta conseguir que tarde un segundo en repetir
57 REM esa instrucción FOR vacía de 50
60 S=S+1
70 IF S=60 THEN S=0:M=M+1
80 IF M=60 THEN M=0:H=H+1
90 LOCATE 10,10
100 PRINT H;":";M;":";S
110 A$=INKEY$
115 REM PULSANDO LA TECLA F SE PARA EL CRONOMETRO
120 IF A$<>"F"AND A$<>"f" THEN GOTO 40
```

TAB(N). Esta función TAB es un poco especial. Debe ir siempre dentro de una instrucción PRINT y significa que lo que se escriba en ese PRINT a continuación de TAB se pondrá en la columna N de la fila en la que esté escribiendo el PRINT.

El siguiente programa es un ejemplo de su uso:

```
10 REM DIBUJO EN PANTALLA
20 REM ******************
30 CLS
40 FOR I=1 TO 15 STEP 2
50 PRINT TAB(10+I);"*"
60 NEXT I
70 FOR I=13 TO 1 STEP -2
80 PRINT TAB(10+I);"*"
90 NEXT I
```

Programa ejemplo de utilización de las funciones alfanuméricas:

```
10 REM CODIGO SECRETO
20 REM **************
22 REM Codifica las palabras convirtiendo cada letra en aquella
23 REM que está 5 posiciones después en el alfabeto
25 CLS
30 INPUT "NUMERO TOTAL DE PALABRAS=",NP
40 DIM N$(NP):DIM C$(NP)
```

```
45 PRINT "INTRODUCE LAS PALABRAS UNA A UNA"
50 FOR I=1 TO NP
60 INPUT N$(I)
65 REM Codificacion de cada letra de la palabra
70 FOR J=1 TO LEN(N$(I))
80 L$=MID$(N$(I),J,1)
90 L=ASC(L$)
100 L=L+5
110 IF L>90 THEN L=L-26
120 C$(I)=C$(I)+CHR$(L)
130 NEXT J
140 NEXT I
145 CLS
150 FOR I=1 TO NP
160 PRINT C$(I);" ";
170 NEXT I
```



ALGUNAS CONSIDERACIONES

En este capítulo se ha tratado de dar una visión de las funciones más importantes y que más se utilizan en BASIC, siendo además aquéllas que son comunes a la gran mayoría de las máquinas que trabajan con BASIC.

De todas formas, se aconseja a los usuarios que consulten en el manual de su ordenador la lista de funciones, porque posiblemente vean que disponen de alguna otra que aquí no se ha especificado o que alguna de las especificadas posee alguna variante para ese equipo. De igual forma, hay algunas de las funciones alfanuméricas aquí expuestas que el SPECTRUM no tiene (LEFT$,RIGHT$,MID$), como podrá observarse al consultar su manual.

También se verán en el manual algunas funciones correspondientes al manejo de gráficos, ficheros o código máquina. Estas funciones se estudiarán en el próximo número que se va a dedicar a estos apartados.

Programas que utilizan funciones:

```
10 REM CALCULO DE LOS DATOS DE UN TRIANGULO RECTANGULO
20 REM *********************************************
30 CLS
40 REM Se suponen conocidos un cateto y un angulo
50 INPUT "VALOR DEL ANGULO(EN GRADOS)";ANG1
55 REM PASO DEL ANGULO A RADIANES
60 ANG1R=ANG1*3.141596/180
70 INPUT "VALOR DEL CATETO";A
80 INPUT "ES EL CATETO CONTIGUO AL ANGULO(S/N)";R$
```

```
90 IF R$="S" THEN H=A/COS(ANG1R)
100 IF R$="N" THEN H=A/SIN(ANG1R)
110 IF R$<>"S" AND R$<>"N" THEN GOTO 80
120 B=SQR(H^2-A^2)
130 ANG2=90-ANG1
140 CLS
150 PRINT "LADOS=";A;B
160 PRINT "HIPOTENUSA=";H
170 PRINT "ANGULOS=";ANG1;ANG2
```

```
10 REM CAIDA ALEATORIA DE 10 ESTRELLAS
20 REM ********************************
30 RANDOMIZE TIMER ' SOLO PARA IBM
40 CLS
50 FOR I=1 TO 10
60 C=INT(RND*32)+1
70 FOR J=2 TO 21
80 LOCATE J,C:PRINT "*"
90 LOCATE J-1,C:PRINT " "
100 NEXT J
110 NEXT I
```

```
10 REM PROGRAMA PARA JUGAR AL MASTER-MIND
20 REM ********************************
30 REM Se debe introducir un numero de 5 cifras con objeto de adivinar
40 REM otro número clave que el programa genera internamente
50 REM el programa dice con cada número introducido las cifras
60 REM que están en el mismo lugar en la clave(aciertos) y las
70 REM que están en distinto lugar(aproximados)
80 RANDOMIZE TIMER 'SOLO PARA IBM
90 DIM CL(5),J(5),EN(5,2)
100 P=1
110 FOR I=1 TO 5
120 CL(I)=INT(RND*10)
130 NEXT I
140 CON=0
150 PRINT "introduce la palabra 'FIN' si quieres terminar y saber la clave"
160 INPUT "INTRODUCE JUGADA";JUG$
170 IF JUG$="FIN" THEN GOTO 440
180 CON=CON+1
190 FOR I=1 TO 5
200 EN(I,1)=1 :EN(I,2)=1
210 J(I)=VAL(MID$(JUG$,I,1))
220 NEXT I
230 AC=0:APR=0
240 FOR I=1 TO 5
250 REM EN(I,1)=0 si se ha encontrado la cifra I de la clave
260 REM jugada.EN(I,2)=0 si se ha encontrado la cifra I de la
270 REM jugada en la clave.
280 IF CL(I)=J(I) THEN AC=AC+1 : EN(I,1)=0 : EN(I,2)=0
290 NEXT I
300 FOR I=1 TO 5
310 FOR K=1 TO 5
320 IF (CL(I)=J(K)) AND EN(I,1) AND EN(K,2) THEN APR=APR+1:EN(I,1)=0:EN(K,2)=0
330 NEXT K :NEXT I
```

```
340 PRINT "ACIERTOS=";AC
350 PRINT "APROXIMADOS=";APR
360 IF AC<5 THEN GOTO 160
370 PRINT "HAS ACERTADO LA COMBINACION EN";CON;"JUGADAS"
380 IF P THEN REC=CON
390 P=0 : IF CON<REC THEN REC=CON
400 PRINT "TU RECORD ES";REC;"JUGADAS"
410 INPUT "QUIERES JUGAR OTRA VEZ?(S/N)",R$
420 IF R$="S" THEN GOTO 110
430 END
440 FOR I=1 TO 5
450 PRINT CL(I);
460 NEXT I
470 END
```

TAMBIEN PODEMOS DEFINIR NUESTRAS PROPIAS FUNCIONES

Hasta ahora hemos visto una serie de funciones predefinidas en BASIC; es decir, una serie de nombres que cuando nosotros los ponemos en un programa, el ordenador realiza las operaciones correspondientes que ya sabe y obtiene un valor como resultado de éstas.

El BASIC nos permite, sin embargo, que nosotros nos inventemos una función. Esto debemos hacerlo con la siguiente instrucción:

DEF FN nombre(arg1,arg2,...)=expresión aritmética

El nombre será uno que nos inventemos nosotros (el que queramos darle a la función que acabamos de definirnos). Entre paréntesis y a continuación del nombre, colocamos los argumentos o datos de entrada que queremos que tenga nuestra función (serán nombres, que cuando queramos que sean alfanuméricos, deberán terminar en $).

Después del signo igual deberemos poner una expresión aritmética según las reglas de las expresiones aritméticas BASIC, que será la operación que realizará nuestra función cuando la ejecutemos. Esta expresión deberá utilizar lógicamente los argumentos que hayamos puesto junto con cualquier otra variable del programa, cualquier operando de los permitidos en el lenguaje o cualquier función de las predefinidas o que hayamos definido nosotros anteriormente (usando su resultado como operando).

Hay algunos ordenadores que precisan poner en la definición las palabras FN y el nombre que le damos, todo junto. Así:

DEF FNnombre(.....

En estos casos, al utilizar la función posteriormente en el programa se debe poner su nombre con las letras FN delante.

Veamos un ejemplo sencillo de definición de una función en un programa y su utilización posterior (que se hace como cualquier función predefinida de las estudiadas). Podemos ver que en este ejemplo la definición se ha realizado según lo expuesto anteriormente.

```
10 REM EJEMPLO DE FUNCION DEFINIDA
20 REM **************************
25 R=1:S=2
30 DEF FNPOL(X)=(X^2)+(X^3)-8
40 PRINT FNPOL(R)
50 LET F=FNPOL(S)
60 PRINT F
70 PRINT FNPOL(0)
```

EJERCICIOS:

1. Hacer un programa que pase un número en base 2 a base decimal. Definir para ello una función que calcule el valor decimal de un dígito binario, teniendo como datos de entrada este dígito y el lugar que ocupa.

2. Hacer un programa para descifrar palabras cifradas en código secreto mediante el programa expuesto en el capítulo.

3. Modificar el programa del cronómetro digital para que después de pararlo pueda volver a ser puesto en marcha mediante otra tecla.

4. ¿Qué se entiende por argumentos de una función?

5. ¿Cuándo decimos que una función es alfanumérica y cómo lo sabemos?

6. Hacer un programa que calcule los divisores de un número cualquiera.

7. Haz un programa que simule el lanzamiento de un dado, pida apuestas a dos personas y diga lo que lleva ganado o perdido cada uno.

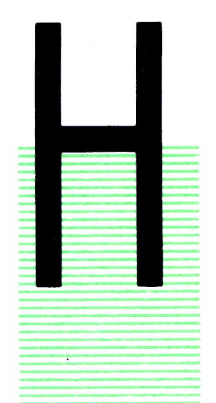
RESPUESTAS:

1.

```
10 REM de base 2 a base 10
20 REM ******************
30 INPUT "numero binario=",B$
40 DEF FNDEC(DIG,P)=DIG*(2^P)
50 FOR I=LEN(B$) TO 1 STEP -1
60 D$=MID$(B$,I,1)
70 D=VAL(D$)
80 N=N+FNDEC(D,LEN(B$)-I)
90 NEXT I
100 PRINT N
```

LAS INSTRUCCIONES READ-DATA-RESTORE 12

INTRODUCCION



HAY programas que antes de comenzar a realizar aquello que deseamos, necesitan tener en sus variables (listas o variables simples), una serie de datos.

Imaginemos el caso, por ejemplo, de un programa que funcione como una agenda telefónica. Cuando le damos orden de ejecución, el programa nos pide un nombre de una persona y a continuación nos escribe por pantalla el teléfono de la persona cuyo nombre le hemos introducido. Para hacer esto, el programa necesita tener previamente en sus variables los datos de los nombres y los teléfonos correspondientes.

Con las instrucciones que hemos visto hasta ahora, tendríamos dos posibilidades de que el programa guardara en sus variables los nombres y los teléfonos; aunque vamos a ver que ninguna de las dos resulta adecuada:

a) La primera sería que al principio del programa nos los pidiera mediante una serie de instrucciones INPUT. Esta solución se cae por su propio peso, ya que si cada vez que ejecutamos el programa tenemos previamente que introducirle los datos que le vamos a pedir, después el programa no nos sirve para nada. Conviene recordar aquí que cada vez que se termina la ejecución de un programa los valores guardados en las variables se destruyen y no se conservan para la siguiente ejecución.

b) Otra solución sería utilizar una instrucción LET para guardar cada dato que queremos en su variable al principio del programa. Esta solución funcionaría de manera correcta, pero en cuanto fueran muchos los datos que quisiéramos introducir habría que poner un gran número de instrucciones LET, con lo que el programa se haría muy largo y farragoso.

La combinación de las instrucciones READ-DATA-RESTORE nos va a permitir introducir en un programa los datos que queramos que tenga guardados en sus variables de una manera muy sencilla.

COMO SE UTILIZAN ESTAS INSTRUCCIONES

Todos los datos que vayamos a querer que un programa tenga guardados en sus variables debemos ponerlos en las instrucciones DATA a continuación de esta palabra y separados por comas. Por ejemplo:

DATA 34,45,567,«garcia»,«PEREZ»

— La instrucción DATA podemos colocarla en cualquier lugar del programa, ya que es una instrucción que no se ejecuta, sino que simplemente almacena una serie de datos que posteriormente va a utilizar el programa.

— Si en una instrucción DATA no nos caben todos los datos que necesitamos poner, basta con que pongamos otra instrucción DATA con un número de instrucción superior a la anterior y en ella sigamos colocando los datos. De la misma manera podemos seguir haciendo lo mismo de manera sucesiva hasta que hayamos metido todos los datos que queríamos, ya que en un programa puede haber tantas instrucciones DATA como queramos.

— Los datos los debemos poner en las instrucciones DATA según el orden en que posteriormente queramos guardarlos en las variables con la instrucción READ, como vamos a ver más adelante. Hay que tener en cuenta que el orden entre varias instrucciones DATA es de menor a mayor número de instrucción.

La instrucción READ debe escribirse poniendo esta palabra y a continuación el nombre de una variable o varias. Cuando se ejecuta una instrucción READ, el programa coge el primer valor que haya en los DATA que aún no haya sido cogido por una instrucción READ y lo guarda en la primera variable que hay puesta después del READ. A continuación hace lo mismo para el resto de las variables en el supuesto de que haya más.

Veamos, como ejemplo, el programa de la agenda telefónica descrito anteriormente con la utilización de las instrucciones READ-DATA.

```
10 REM AGENDA TELEFONICA
20 REM *****************
30 CLS
40 DIM D(5)
50 DIM N$(5)
60 DIM T(5)
70 DATA "JUAN PEDRO",2000011,"CARLOS",4655555,"GONZALO",2222222
80 DATA "PALOMA",7684536,"ESTER",7777755
90 CLS
100 FOR X=1 TO 5
110 READ N$(X):READ T(X)
120 NEXT X
130 INPUT "TECLEA EL NOMBRE";P$
```

```
140 FOR X=1 TO 5
150 IF P$=N$(X) THEN PRINT T(X)
160 NEXT X
170 PRINT :PRINT "QUIERES SABER MAS NUMEROS  (S/N)?"
180 A$=INKEY$
190 IF A$="" THEN GOTO 180
200 IF A$="S" THEN GOTO 130
```

Como hemos dicho, cada vez que el programa ejecuta una instrucción READ, lo que hace es coger el siguiente dato del DATA al último que cogió en la última instrucción READ y guardarlo en la variable que haya después del READ. Si el tipo de esta variable no coincide con el tipo del dato correspondiente del DATA el programa se dejará de ejecutar y emitirá un mensaje de error. Por ello, al utilizar instrucciones READ-DATA hay que cuidar que el orden en que se introducen los datos vaya a ser el mismo que el orden en que luego se van a leer. En el ejemplo anterior se observa que los datos se leen dentro de un bucle FOR, y en él, cada vez que se repite, se lee un nombre y se guarda en su variable de la lista correspondiente, y a continuación se lee su teléfono y se guarda en la variable con el mismo número de la lista de los teléfonos. Por tanto, este orden de lectura nos exige que en las instrucciones DATA introduzcamos los datos siguiendo el orden de nombre y a continuación su número de teléfono para todas las personas.



LA INSTRUCCION RESTORE

La instrucción RESTORE se escribe con el formato:

RESTORE número de instrucción

El número de instrucción deberá corresponder con una instrucción DATA del programa y cuando se ejecute una instrucción RESTORE lo que hace el programa es que a partir del próximo READ se comenzarán a coger datos del DATA cuyo número estaba en la RESTORE, siguiéndose, cuando se acabe éste, con los DATA que haya a continuación de él hasta que se ejecute otra RESTORE.

En el siguiente ejemplo podemos observar el funcionamiento de estas tres instrucciones:

```
10 REM EJEMPLO DATA-READ RESTORE
20 REM**************************
30 DATA 1,2
```

```
40 DATA 3
50 READ A,B,C
60 RESTORE 40
70 READ D
80 READ N
90 DATA "MARZO"
```

La primera READ toma para sus variables los valores 1, 2 y 3.

A continuación se ejecuta la instrucción 60, que hace que a partir de ahí se comiencen a leer los datos de la DATA de 40.

La instrucción 70 guarda el 3 en la variable del READ y la 80 genera un error y detiene el programa porque toca leer el dato «MARZO» y no se puede guardar en la variable numérica N.

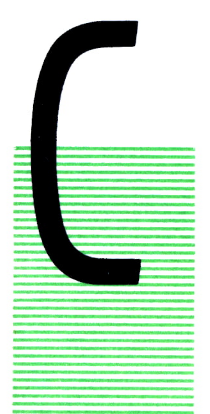
EJERCICIOS:

1. ¿Qué parecido tienen y en qué se diferencian las instrucciones INPUT y READ?

2. Hacer un programa que a partir de los datos de la extensión y habitantes de una serie de países nos diga qué país es más grande y cuál tiene más habitantes entre tres que nosotros le introduzcamos por pantalla.

UNA INSTRUCCION QUE NOS FACILITA LAS COSAS: ON..GOTO 13

INTRODUCCION



UANDO estudiamos la instrucción IF, veíamos que servía para realizar una instrucción o un conjunto de instrucciones únicamente en el caso de que se cumpliera una determinada condición.

Hay ocasiones, sin embargo, en que queremos que el programa haga una serie de acciones dependiendo de unas condiciones que no toman únicamente dos valores (sí o no). Imaginemos el caso de que en una variable tenemos guardado un número comprendido entre 1 y 10 y queremos que un programa nos escriba el valor que tenemos en la variable con letras. La condición en este caso para que el programa haga una instrucción u otra será el valor que haya guardado en la variable.

Con lo que hemos visto hasta ahora, tendríamos que escribir el programa de la siguiente manera:

```
IF N=1 THEN PRINT «UNO»
IF N=2 THEN PRINT «DOS»
IF N=3 THEN PRINT «TRES»
........................
```

y así tendríamos que seguir sucesivamente tantas veces como distintas condiciones puedan darse para las que queramos realizar alguna instrucción (en este caso la instrucción de escribir un mensaje, distinto cada vez).

En este capítulo vamos a ver la instrucción ON..GOTO que nos permite solucionar este tipo de programas de una manera mucho menos pesada.

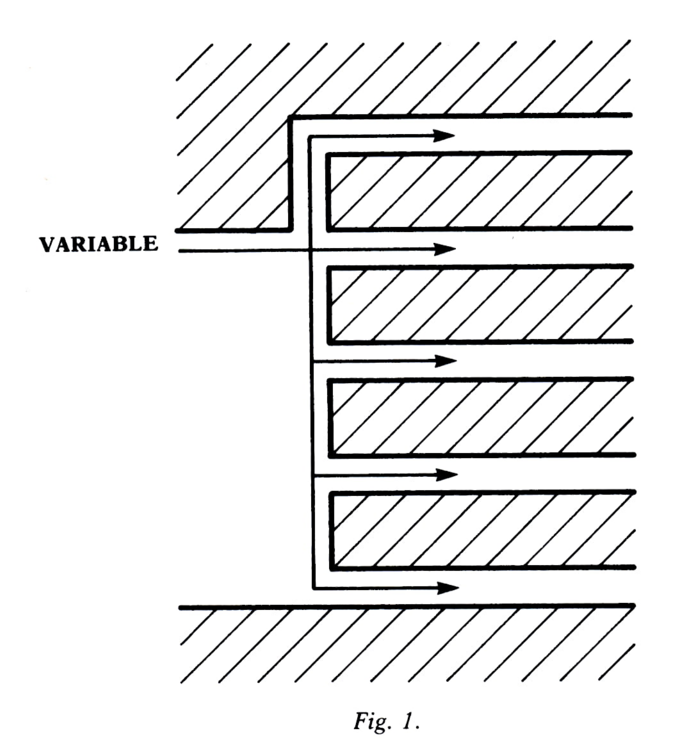


Fig. 1.



LA INSTRUCCION ON..GOTO

La instrucción ON..GOTO tiene el siguiente formato:

ON variable GOTO N1,N2,N3,...

N1, N2, N3,... son números de instrucciones del programa. Cuando se ejecuta una instrucción ON..GOTO el programa hace lo siguiente:

— Si en la variable hay guardado un 1, el programa salta a la instrucción cuyo número está en primer lugar a continuación de la palabra GOTO (N1).

— Si en la variable hay guardado un 2, el programa salta a la instrucción cuyo número está en segundo lugar a continuación de la palabra GOTO.

— Si en la variable hay un 3, salta a la instrucción cuyo número está en tercer lugar, y así sucesivamente.

En el caso de que no haya ningún número de instrucción en el lugar correspondiente al número que hay en la variable, el programa continuará por la instrucción que haya a continuación del ON..GOTO.

Veamos cómo quedaría el programa que nos resolviera el problema planteado anteriormente de escribir con letras los números guardados en una variable:

```
10 REM EJEMPLO DE ON .. GOTO
20 REM ********************
30 INPUT N
40 ON N GOTO 60,70,80,90,100,110,120,130,140,150,160,160,160,160,160
50 PRINT "MAYOR QUE 15":GOTO 170
60 PRINT "UNO":GOTO 170
70 PRINT "DOS":GOTO 170
80 PRINT "TRES":GOTO 170
90 PRINT "CUATRO":GOTO 170
100 PRINT "CINCO":GOTO 170
110 PRINT "SEIS":GOTO 170
120 PRINT "SIETE":GOTO 170
130 PRINT "OCHO":GOTO 170
140 PRINT "NUEVE":GOTO 170
150 PRINT "DIEZ":GOTO 170
160 PRINT "ESTA ENTRE 11 Y 15"
170 REM Aquí continuaría el programa
```

En la instrucción ON..GOTO cuando N valga 1, el programa saltará a la instrucción 60 (colocada en primer lugar), cuando N valga 2, saltará a la 70, y así sucesivamente. Cuando N valga 11, 12, 13, 14 ó 15, saltará a la misma instrucción (160), ya que para estos cinco valores queremos que el programa haga lo mismo (que escriba que está entre esos valores). Cuando N valga más de 15, como no hay ningún número de instrucción que le corresponda, seguirá por la siguiente al ON..GOTO (50), escribiendo en ella lo que queremos para el caso de que sea mayor que 15.

Esta instrucción ON..GOTO no existe en el SPECTRUM. Se puede realizar una estructura equivalente siempre que la diferencia entre los números de instrucción colocados después del GOTO sea la misma. En el ejemplo anterior podríamos realizarlo siempre que supiésemos que el número N va a estar seguro entre 1 y 10, de la siguiente manera:

GOTO (I*10)+50

Sustituyendo el ON..GOTO por esta instrucción podemos comprobar que el programa realiza lo mismo siempre que el número esté comprendido entre 1 y 10. El resto de los casos realizados con la ON..GOTO en el SPECTRUM no habría otra solución que hacerlos mediante instrucciones IF.

Como ejemplo final, ponemos a continuación un programa que utiliza gran parte de las instrucciones BASIC estudiadas en este libro. Se trata de un programa que realiza un juego consistente en adivinar una serie de palabras a partir de su definición y la letra inicial y final. El programa irá pi-

diendo letras y aquellas que estén en la palabra, las irá poniendo hasta que se complete ésta.

Las palabras que hay que adivinar se meten en las instrucciones DATA mediante los códigos ASCII de cada una de las letras para que así no puedan descubrirse al observar el listado. El programa, al leer estos números, los transforma en su letra correspondiente. La palabra es elegida al azar por el programa entre las 20 que hay, y mediante la instrucción ON..GOTO se selecciona su definición.

```
10 REM       ********************************************************
20 REM       **                                                    **
30 REM       **       PROGRAMA PARA ACERTAR NOMBRES DE PLANTAS A   **
40 REM       **       PARTIR DE UNA DEFINICION UTILIZANDO EL JUEGO **
50 REM       **       CONSISTENTE EN IR ADIVINANDO LAS LETRAS      **
60 REM       **                                                    **
75 REM       **    ESTE PROGRMA NO ES VALIDO PARA SPECTRUM        **
80 REM       ********************************************************
90 DIM P$(20,11):DIM L$(11)
95  REM LA INSTRUCCION 100 SOLO ES NECESARIO UTILIZARLA PARA MS-DOS (IBM O
        COMPATIBLES)
100 RANDOMIZE TIMER
110 DATA 84,82,73,71,79,0,67,69,66,65,68,65,0,65,86,69,78,65,0,71,65,82,66,65,78
120 DATA 90,79,83,0,74,85,68,73,65,83,0,71,85,73,83,65,78,84,69,83,0,65,76,67,65
130 DATA 67,72,79,70,65,0,84,79,77,65,84,69,0,90,65,78,65,72,79,82,73,65,0,78,65
140 DATA 82,65,78,74,65,0,76,73,77,79,78,0,80,79,77,69,76,79,0,77,69,76,79,67,79
150 DATA 84,79,78,0,67,69,82,69,90,65,0,65,66,69,84,79,0,82,79,66,76,69,0,65,66
160 DATA 69,68,85,76,0,80,69,78,83,65,77,73,69,78,84,79,0,79,82,81,85,73,68,69
170 DATA 65,0,65,77,65,80,79,76,65,0
180 PRINT :PRINT "¡UN MOMENTO!,ESTOY CARGANDO LOS DATOS"
190 FOR I=1 TO 20
200 LET J=1
210 READ A
220 IF A=0 THEN FOR K=J TO 11 :LET P$(I,K)=" ":NEXT K :GOTO 250
230 LET P$(I,J)=CHR$(A)
240 LET J=J+1:GOTO 210
250 NEXT I
260 LET PAL=INT(RND(1)*20)+1
270 FOR J=1 TO 11 :LET L$(J)=" ":NEXT J
280 LET FT=0 :LET CON=0
290 CLS
300 ON PAL GOTO 310,310,310,320,320,320,330,340,350,360,360,360,370,380,390,400,
            410,420,420,430
310 PRINT "CEREAL CULTIVADO AMPLIAMENTE EN TODA EUROPA":GOTO 440
320 PRINT "LEGUMBRE CULTIVADA EN LOS PAISES PROXIMOS AL MEDITERRANEO Y A LA
          COSTA ATLANTICA":GOTO 440
330 PRINT "HORTALIZA       ":GOTO 440
340 PRINT "HORTALIZA EMPLEADA EN LA CONFECCION DE ENSALADAS":GOTO 440
350 PRINT "HORTALIZA QUE CRECE BAJO TIERRA":GOTO 440
360 PRINT "FRUTA TIPICA DE ZONAS MEDITERRANEAS DE LA QUE SE OBTIENEN EXCELENTES
          ZUMOS":GOTO 440
370 PRINT "FRUTA          ":GOTO 440
380 PRINT "FRUTA DE COLOR ROJIZO EMPLEADA EN REPOSTERIA":GOTO 440
390 PRINT "ARBOL DE HOJA PERENNE UTILIZADO COMO ARBOL TIPICO NAVIDEÑO":GOTO 440
400 PRINT "ARBOL DE HOJA CADUCA MUY EXTENDIDO EN TODOS LOS BOSQUES EUROPEOS":
          GOTO 440
410 PRINT "ARBOL CUYA SAVIA ES UTILIZADA EN ALGUNOS PAISES PARA PREPARAR
          BEBIDAS":GOTO 440
420 PRINT "PLANTA QUE DESTACA POR LA BELLEZA DE SUS FLORES":GOTO 440
430 PRINT "FLOR CONOCIDA POR SU BRILLANTE COLOR ROJO"
440 LET L$(1)=P$(PAL,1)
450 FOR I=11 TO 2 STEP-1
460 IF P$(PAL,I)<>" " THEN LET L$(I)=P$(PAL,I):LET F=I:GOTO 480
470 NEXT I
```

```
480 LOCATE 10,10
490 FOR I=1 TO F
500 PRINT L$(I);" ";
510 NEXT I
520 PRINT:PRINT TAB(12);
530 FOR I=2 TO F-1
540 PRINT"- ";
550 NEXT I
555 REM PARA AMSTRAD :560 LOCATE 1,15
560 LOCATE 15,1
570 IF FT THEN GOTO 690
580 LET FT=1
590 PRINT "TECLEA LAS LETRAS QUE CREAS QUE FALTEN EN LA PALABRA(MAYUSCULAS)"
600 LET A$=INKEY$
610 IF A$="" THEN GOTO 600
620 FOR I=2 TO F-1
630 IF P$(PAL,I)=A$ THEN LET L$(I)=P$(PAL,I):LET LT=1
640 IF L$(I)=" " THEN LET FT=0
650 NEXT I
660 IF LT=0 THEN CON=CON+1 :PRINT:PRINT"LO SIENTO,LA LETRA ";A$;" NO ESTA"
670 LET LT=0
680 GOTO 480
690 PRINT "                                                              "
700 PRINT :PRINT "                                                  "
710 PRINT "¡ENHORABUENA!,LA HAS ACERTADO":PRINT
720 PRINT "HAS TECLEADO ";CON;" LETRAS ERRONEAS AL INTENTAR ACERTAR LA PALABRA"
730 PRINT "QUIERES JUGAR OTRA VEZ? (S/N)"
740 LET A$=INKEY$
750 IF A$="S" THEN GOTO 260
760 IF A$<>"N" THEN GOTO 740
780 REM ****************************************************
790 REM **   MODIFICACIONES PARA COMMODORE :              **
800 REM **  290 PRINT CHR$(147)                           **
810 REM **  480 FOR I=1 TO 8                              **
820 REM **  482 PRINT                                     **
830 REM **  484 NEXT I :PRINT TAB(10);                    **
840 REM **  560 FOR I=1 TO 4                              **
850 REM **  563 PRINT                                     **
860 REM **  565 NEXT I                                    **
870 REM **  600 GET A$                                    **
880 REM **  680 GOTO 290                                  **
890 REM **  740 GET A$                                    **
900 REM ****************************************************
```

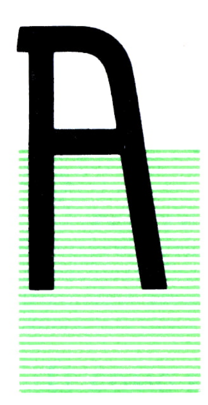
EJERCICIOS:

1. Hacer un programa que pida dos operandos y a continuación escriba por pantalla:

SUMA	*-->1*
RESTA	*-->2*
MULTIPLICACION	*-->3*
DIV. ENTERA	*-->4*
DIV. NORMAL	*-->5*
RESTO DIVISION	*-->6*

Después de escribir esto, el programa pedirá un número del 1 al 6, y según el número introducido, realizará con los dos operandos la operación correspondiente, escribiendo su resultado. Si el número introducido no está entre 1 y 6, volverá a escribir las operaciones por pantalla y pedirá un nuevo número.

CONCLUSION 14



N este primer libro dedicado a estudiar el lenguaje de programación BASIC se ha tratado de dar una idea general de los problemas que se pueden plantear, las órdenes que necesitamos dar a nuestro computador para que realice aquello que deseamos, y hemos visto cómo se puede conseguir esto en un lenguaje de programación BASIC.

Se han estudiado las instrucciones principales que se necesitan.

El siguiente tomo se dedicará al estudio de otros aspectos importantes que puede hacer el ordenador a través de las instrucciones de un programa BASIC, como pueden ser el manejo de ficheros, la realización de gráficos o la resolución de problemas típicos que se nos plantean en la vida real.

Esperamos que el lector se haya dado cuenta tanto de las ventajas como de los inconvenientes que nos plantea el programar en BASIC. El BASIC es un lenguaje sencillo en el aspecto que no necesita declarar las variables que utiliza, como ocurre en otros lenguajes, y las instrucciones tienen el formato más sencillo que puede existir. Sin embargo, a la hora de realizar programas complejos, resulta mas complicado por no permitir estructurar las distintas partes del programa en bloques, como ocurre con otros lenguajes, siendo de ellos el más representativo el PASCAL.

Como ya decíamos en el segundo capítulo, es importante destacar, sin embargo, que el BASIC es un lenguaje tan potente como cualquier otro y que con él podemos realizar cualquier operación que queramos que haga nuestro ordenador.

De igual manera, hay algunas aplicaciones, como la resolución de problemas matemáticos o la realización de tareas administrativas que no requieran el manejo de muchos datos, para las que el BASIC resulta más sencillo que otros lenguajes de programación.

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

INDICE GENERAL

5 AÑADA PERIFERICOS A SU ORDENADOR

Breve descripción de varios periféricos que facilitan la comunicación con el ordenador personal, con algunos ejemplos de fácil construcción: ratón, lápiz óptico, marco para pantalla táctil...

6 GRAFICOS ANIMADOS CON EL ORDENADOR

En este libro las técnicas utilizadas para la animación son el resultado de unas pocas ideas básicas muy sencillas de comporender. Descubrirá los trucos y secretos de movimientos, choques, rebotes, explosiones, disparos, saltos, etc.

7 JUEGOS INTELIGENTES EN MICROORDENADORES

Los ordenadores pueden enfrentarse de forma «inteligente» ante puzzles y otros tipos de juegos. Esto es posible gracias al nuevo enfoque que ha dado la IA a la tradicional teoría de juegos.

8 PERIFERICOS INTERACTIVOS PARA SU ORDENADOR

Descripción detallada de la forma de construir, paso a paso y en su propia casa, dispositivos electrónicos que aumentarán la potencia y facilidad de uso de su ordenador: tableta digitalizadora, convertidores de señales analógicas, comunicaciones entre ordenadores.

9 COMO HACER DIBUJOS TRIDIMENSIONALES EN EL ORDENADOR PERSONAL

Compruebe que también con su ordenador personal puede llegar a diseñar y calcular imágenes en tres dimensiones con técnicas semejantes a las utilizadas por los profesionales del dibujo con equipos mucho más sofisticados.

10 PRACTIQUE MATEMATICAS Y ESTADISTICA CON EL ORDENADOR

En este libro se repasan los principales conceptos de las Matemáticas y la Estadística, desde un punto de vista eminentemente práctico y para su aplicación al ordenador personal. Se basan los diferentes textos en la presentación de pequeños programas (que usted podrá introducir en su ordenador personal).

11 CRIPTOGRAFIA: LA OCULTACION DE MENSAJES Y EL ORDENADOR

En este libro se presentan las técnicas de mensajes a través de la criptografía desde los primeros tiempos hasta la actualidad, en que el uso de los computadores ha proporcionado la herramienta necesaria para llegar al desarrollo de esta técnica.

12 APL: LENGUAJE PARA PROGRAMADORES DIFERENTES

APL es un lenguaje muy potente que proporciona gran simplicidad en el desarrollo de programas y al mismo tiempo permite programar sin necesidad de conocer todos los elementos del lenguaje. Por ello es ideal para quienes reúnan imaginación y escasa formación en Informática.

13 ECONOMIA DOMESTICA CON EL ORDENADOR PERSONAL

Breve introducción a la contabilidad de doble partida y su aplicación al hogar, con explicaciones de cómo utilizar el ordenador personal para facilitar los cálculos, mediante un programa especialmente diseñado para ello.

14 COMO SIMULAR CIRCUITOS ELECTRONICOS EN EL ORDENADOR

Introducción a los diferentes métodos que se pueden emplear para simular y analizar circuitos electrónicos, mediante la utilización de diferentes lenguajes.

15 COMO CONSTRUIR SU PROPIO ORDENADOR

Cuando se trabaja con un ordenador, lo único que puede apreciarse, a simple vista, es una especie de caja negra que, misteriosamente, acepta una serie de instrucciones. En realidad, un ordenador es una máquina capaz de recibir, transformar, almacenar y suministrar datos.

16 EL ORDENADOR COMO INSTRUMENTO MUSICAL Y DE COMPOSICION

Análisis de cómo se puede utilizar el ordenador para la composición o interpretación de música. Libro eminentemente práctico, con numerosos ejemplos (que usted podrá practicar en su ordenador casero) y lleno de sugerencias para disfrutar haciendo de su ordenador un verdadero instrumento musical.

17 SISTEMAS OPERATIVOS: EL SISTEMA NERVIOSO DEL ORDENADOR

Características de diversos sistemas operativos utilizados en los ordenadores personales y caseros. Se trata de llegar al conocimiento, ameno aunque riguroso, de la misión del sistema operativo de su ordenador, para que usted consiga sacar mayor rendimiento a su equipo.

18 UNIX, EL ESTANDAR DE LOS SISTEMAS OPERATIVOS MULTIUSUARIO

La aparición y posterior difusión del sistema operativo UNIX supuso una revolución en el mercado, de tal modo que se ha convertido en el estándar de los sistemas multiusuario. Su aparente complejidad podría provocar, en principio, un primer rechazo, pero debido a su potencia se convierte rápidamente en una extraordinaria herramienta de trabajo apta para cualquier tipo de aplicaciones.

19 EL ORDENADOR Y LA ASTRONOMIA

Los cálculos astronómicos y el conocimiento del firmamento en un libro apasionante y curioso.

20 VISION ARTIFICIAL. TRATAMIENTO DE IMAGENES POR ORDENADOR

El procesado de imágenes es un campo de reciente y rápido desarrollo con importantes aplicaciones en área tan diversas como la mejora de imágenes biomédicas, robóticas, teledetección y otras aplicaciones industriales y militares. Se presentan los principios básicos, los sistemas y las técnicas de procesado más usuales.

21 PRACTIQUE HISTORIA Y GEOGRAFIA CON SU ORDENADOR

Libro interesante para los aficionados a estas ciencias, a quienes presenta una nueva visión de cómo utilizar el microordenador en su estudio.

22 LA CREATIVIDAD EN EL ORDENADOR. EXPERIENCIAS EN LOGO

El LOGO es un lenguaje enormemente capacitado para la creación principalmente gráfica y en especial para los niños. En este sentido se han desarrollado numerosas experiencias. En el libro se analizan estas experiencias y las posibilidades del LOGO en este sentido, así como su aplicación a su ordenador casero para que usted mismo (o con sus hijos) pueda repetirlas.

23 EL LENGUAJE C, PROXIMO A LA MAQUINA

Lenguaje de programación que se está imponiendo en los microordenadores más grandes, tanto por su facilidad de aprendizaje y uso, como por su enorme potencia y su adecuación a la programación estructurada. Vinculado íntimamente al sistema operativo UNIX es uno de los lenguajes de más futuro entre los que se utilizan los micros personales.

24 BASIC

El lenguaje BASIC, es la forma más fácil de aprender las instrucciones más elementales con las que podemos mandar a nuestro ordenador que haga las más diversas tareas.

25 COMO ELEGIR UNA HOJA ELECTRONICA DE CALCULO

En este título se estudian las diferentes versiones existentes de esta aplicación típica, desde el punto de vista de su utilidad para, en función de las necesidades de cada usuario y del ordenador de que dispone, poder elegir aquélla que más se adecúe a cada paso.

26 PRACTIQUE FISICA Y QUIMICA CON SU ORDENADOR

Libro eminentemente práctico para realizar pequeños «experimentos» con su ordenador y distraerse de un modo útil.

27 EL ORDENADOR Y LA LITERATURA

En este libro se examinan procesadores de textos, programas de análisis literario y una curiosa aplicación desarrollada por el autor: APOLO, un programa que compone estructuras poéticas.

28 PRACTIQUE CIENCIAS NATURALES CON EL ORDENADOR

Ejemplos sencillos para practicar con el ordenador. Casos curiosos de la Naturaleza en forma de programas para su ordenador personal.

29 ERGONOMIA: COMUNICACION EFICIENTE HOMBRE-MAQUINA

Análisis de la comunicación entre el hombre y la máquina, y estudio de diferentes soluciones que tienden a facilitarla lo más posible.

30 LOS LENGUAJES DE LA INTELIGENCIA ARTIFICIAL

Libro en que se describen los lenguajes específicos para la «elaboración del saber» y los entornos de programación correspondientes. El conocimiento de estos lenguajes, además de interesante en sí mismo, es sumamente útil para entender todo lo que la Informática Artificial supondrá para el futuro de la Informática.

31 ¿MAQUINAS MAS EXPERTAS QUE LOS HOMBRES?

Después de situar los «sistemas expertos» en el contexto de la inteligencia artificial y describir su construcción, su funcionamiento, su utilidad, etc., se analiza el papel que pueden tener en el futuro (y presente, ya) de la Informática.

NOTA:
Ediciones Siglo Cultural, S. A., se reserva el derecho de modificar, sin previo aviso, el orden, título o contenido de cualquier volumen de esta colección.

Lo más importante al ponerse delante de un ordenador es saber qué hay que hacer para que realice aquello que nosotros deseamos.

El lenguaje BASIC es la forma más fácil de aprender las instrucciones más elementales con las que podemos mandar a nuestro ordenador que haga las más diversas tareas.

Convierta su ordenador en un instrumento de su trabajo y de sus necesidades.